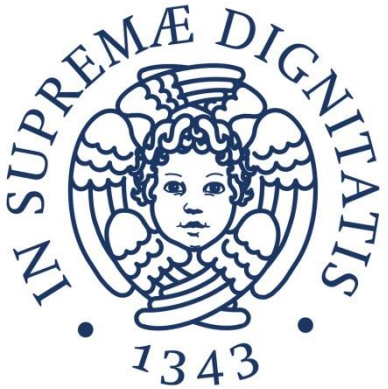
UNIVERSITA DEGLI STUDI DI PISA



Facoltà di ingegneria

Corso di Laurea Specialistica In Ingegneria Informatica

Definizione di strumenti e analisi di gerarchie di memoria per sistemi mono e multiprocessore on-chip

Candidato:
David Serena

Relatori:
Pierfrancesco Foglia
Alessio Bechini

A tutti coloro che mi hanno sostenuto

in tutti questi anni…

SOMMARIO

In questo documento verranno analizzate varie tecnologie di memorie gerarchiche on chip per sistemi mono e multiprocessore ,al fine di compararne le prestazioni e i consumi, con particolare interesse per le tipologie NUCA (Non Uniform Cache Access).

Verranno inoltre presentati tutti gli strumenti utilizzati per effettuare tali valutazioni. Punteremo la nostra attenzione sul simulatore *Sim-Alpha* , utilizzato per la simulazione e l'analisi delle memorie e ne descriveremo le modifiche strutturali apportate , per supportare le nuove features necessarie per eseguire le valutazioni presenti nel documento.

INDICE

INTRODUZIONE

I processori ad alte prestazioni attualmente presenti in commercio , incorporano al loro interno una grande quantità di memoria cache di livello 2 (L2) e di livello 3 (L3). Visto i continui progressi nel campo litografico e la conseguente riduzione del processo produttivo utilizzato, le dimensioni delle memorie cache sono destinate ad aumentare in futuro, poiché una tecnologia più piccola consente una maggiore concentrazione di bit per mm^2 .Questo aumento di memoria , se da una parte contribuisce ad un miglioramento delle prestazioni per l'esecuzione dei processi , in quanto diminuiscono gli accessi in RAM , da una altra parte , introduce dei problemi di latenza di accesso. Le Cache monolitiche di grande dimensione introducono una latenza di accesso uniforme su tutta la cache e vengono chiamate UCA (Uniform Cache Architecture). Per cache di piccole dimensioni, tempi di accesso uniformi ai dati non introducono ritardi considerevoli che non inficiano le prestazioni , o ne contribuiscono in minima parte tanto da considersi trascurabile, ma aumentando la quantità di menoria , questi ritardi di collegamento , possono degradare le prestazioni ,fino a vanificare il contributo dell'aumento di memoria.Per questo motivo sono state introdotte architetture di cache ad accesso non unorme , chiamate NUCA (Non –Uniform Cache Access)

Esistono due tipologie di cache NUCA , la prima chiamata S-NUCA ,presenta un tipodi mapping statico mentre la seconda, chiamata D-NUCA consente invece un mappaggio dinamico spostando i dati più acceduti vicino al processore, dove sono presenti i bachi ad accesso più veloce.

La tesi in corso consiste nell'analizzare queste due tipologie di cache sia sotto l'aspetto delle prestazione che dei consumi , e allo stesso tempo affinaregli strumenti per effetuare tali indagini .

In particolare verranno presentati i seguenti argomenti

MODIFICHE AL SIMUALTORE SIM-ALPHA

1.1. Introduzione al simulatore

Il simulatore sim-alpha è un progetto open source , derivato da SimpleScalar , che simula il processore Alpha AXP, un'architettura di processori di tipo RISC sviluppata e prodotta dalla Digital Equipment Corp (DEC). Il software , scritto completamente in C, offre la possibilità di simulare i benchmark della suite SPEC (Standard Performance Evaluation Corporation), associazione no profit per la scelta , il mantenimento e l'evoluzione di stress-test per CPU, memorie onchip e compilatori. I vari benchmark, reperibili in formato EIO (External Input/Output) , offrono la possibilità di testare in maniera intensiva i componenti delle CPU, per poterne valutare le prestazioni e i consumi in condizioni estreme di utilizzo.

La prima versione del simulatore è stata realizzata da Stephen W. Keckler dell'università del texas (Austin) e nel corso degli anni il simulatore è stato modificato , aggiungendo features , con lo scopo di simulare più tipologie di cache ,anche in modalità multiprocessore , per venire incontro alle esigenze sempre più esigenti delle applicazioni odierne. Il lavoro descritto in seguito illustra le modifiche apportate al simulatore per poter inserire alcune funzionalità necessarie affinare l'analisi delle prestazioni delle cache o per valutarne i consumi. Come vedremo le principali modifiche hanno riguardato l'implementazione delle seguenti caratteristiche:

- Supporto multiprocessore alle cache di tipo S-NUCA
- Estensione delle tecniche di risparmio energetico alla nuova tipologia di cache
- Strumento per la misurazione della potenza istantanea

Durante l'esposizione del lavoro faremo riferimento esplicito al concetto di modulo software che, per definizione, nel linguaggio C è definito come l'insieme di due file :

- ***nome_modulo.h***
- ***nome_modulo.c***

il primo file(.h) contiene gli header che contiene solo le definizioni di funzioni e tipi utilizzabili da altri moduli, mentre il secondo (.c) contiene l'implementazione del modulo stesso.

1.2. Introduzione del supporto per le cache di tipo S-NUCA

La principale modifica apportata al simulatore , riguarda l'introduzione del supporto per le cache di tipo S-NUCA in ambiente multi programmato. L'idea di fondo seguita nello svolgimento di questo task ,consiste nel creare un unico strumento per l'esecuzione di simulazioni che supporti vari tipi di cache , in modo da unificare i futuri sforzi implementativi su un singolo software e semplificare la gestione delle simulazioni stesse.

La necessità di avere un simulatore multiprocessore unico , che sappia gestire entrambe le tipologie di cache NUCA (S-NUCA e D-NUCA),oltre alla cache normale cache plain UCA ,ha portato all'unificazione dei vecchi simulatori specifici per le singole cache in nuovo simulatore. È stato modificato il simulatore "sim-alpha_d-nuca_cmp" ,che in precedenza gestiva solamente le cache D-NUCA e UCA, inserendo il supporto per la cache S-NUCA(supportate in precedenza solo in un altro simulatore , solo monoprocessore) ed estendendole per la gestione multiprocessore.Per ottenere questo è stato creato un fork del precedente simulatore "sim-alpha_d-nuca_cmp" , resettando il numero di versione e aggiungendo le nuove features . Il risultato è il nuovo simulatore chiamato "sim-alpha_nuca_cmp"

Per rendere possibile la coesistenza di più tipologie di cache nello stesso simulatore, l'organizzazione della cache all'interno del simulatore è stata fortemente modificata. L'idea seguita consiste nel creare dei moduli software per ciascuna cache ,che contengono l'implementazione specifica della particolare tecnologia e il cui interfacciamento con il sistema deve avvenire per mezzo di API standard. La scelta della tipologia di cache da utilizzare deve essere effettuata all'avvio del programma , in maniera dinamica. In questo modo si ottiene un simulatore unico per molteplici tipologie di cache semplificando la gestione delle simulazione e facilitando le future modifiche , evitando così inutili frammentazione del codice tra i vari simulatori. Per facilitare il meccanismo di selezione , si provvederà a creare uno specifico parametro letto in fase di configurazione del simulatore. In seguito verranno mostrate sia , l'organizzazione dei file che implementano la cache , sia le nuove opzioni che sono state introdotte per effettuare il meccanismo di scelta.

1.2.1. Riorganizzazione dell'architettura della cache nel simulatore

In origine ,l'architettura dei moduli che simulavano il comportamento temporale delle cache consisteva in un unico modulo software , che gestiva le varie tipologie di cache allora implementate(D-NUCA e UCA). Al suo interno , le funzioni che lo componevano , implementavano tutte le varie tipologie di cache, e la scelta di che blocchi di codice eseguire , per la cache in uso, era fatta per mezzo di costrutti "if" all'interno del codice. Questo approccio, risulta poco gestibile e di difficile modifica , in quanto il codice tende ad allungarsi in maniera incontrollabile , con funzioni mastodontiche che limitano moltissimo la capacità di comprensione del funzionamento , e complicano notevolmente l'opera di debug. Tale architettura è mostrata in figura , dove si può notare l'unico modulo ,chiamato *cache_timing* ,che gestisce la simulazione termporale. Si possono notare alcuni moduli di supporto che implementano alcune funzionalità specifiche, in particolare il modulo *nuca* che si occupa della gestione dei banchi d-nuca. Per sopperire a gli inconvenienti di un architettura monolitica , descritti in precedenza , si è deciso di ristrutturare radicalmente l'implementazione delle cache ,al fine di passare un architettura modulare che faciliti la gestione e le future modifiche dei moduli che compongono tale subsistema. Per questo motivo è stato creato un modulo software per ciascuna delle tipologie di cache implementate , più alcuni moduli di supporto per la gestione delle funzioni e strutture dati comuni a ciascuna cache. La selezione del modulo da usare durante l'esecuzione è fatta in fase di configurazione del simulatore stesso , durante la lettura dei parametri.

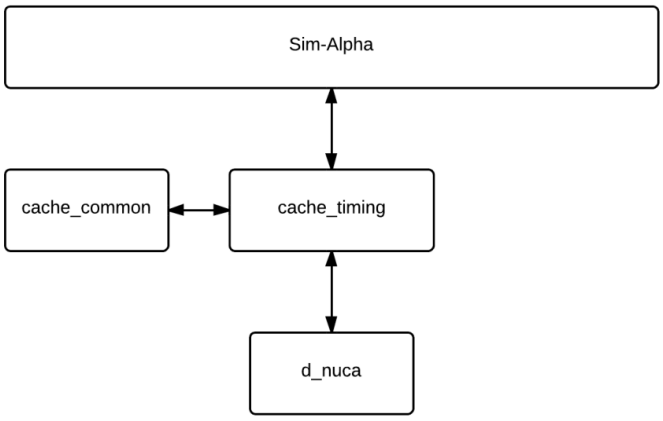


Figura 1 : Vecchia architettura delle cache all'interno del simulatore sim-alpha

Questa scelta è motivata dal fatto che la tipologia di memoria utilizzata è fissa durante tutto l'arco della simulazione, ed è fissata proprio dai parametri con cui il simulatore viene lanciato. Per questo motivo ha senso definire in fase di configurazione il modulo usato , in quanto non sarà modificato per tutta l'esecuzione della simulazione. Uno schema logico della nuova organizzazione della cache è presentato in figura , dove si può notare che i vari moduli che implementato le varie tipologie di cache sono derivati dal modulo generico che ne definisce anche l'interfaccia per l'utilizzo all'interno del simulatore.

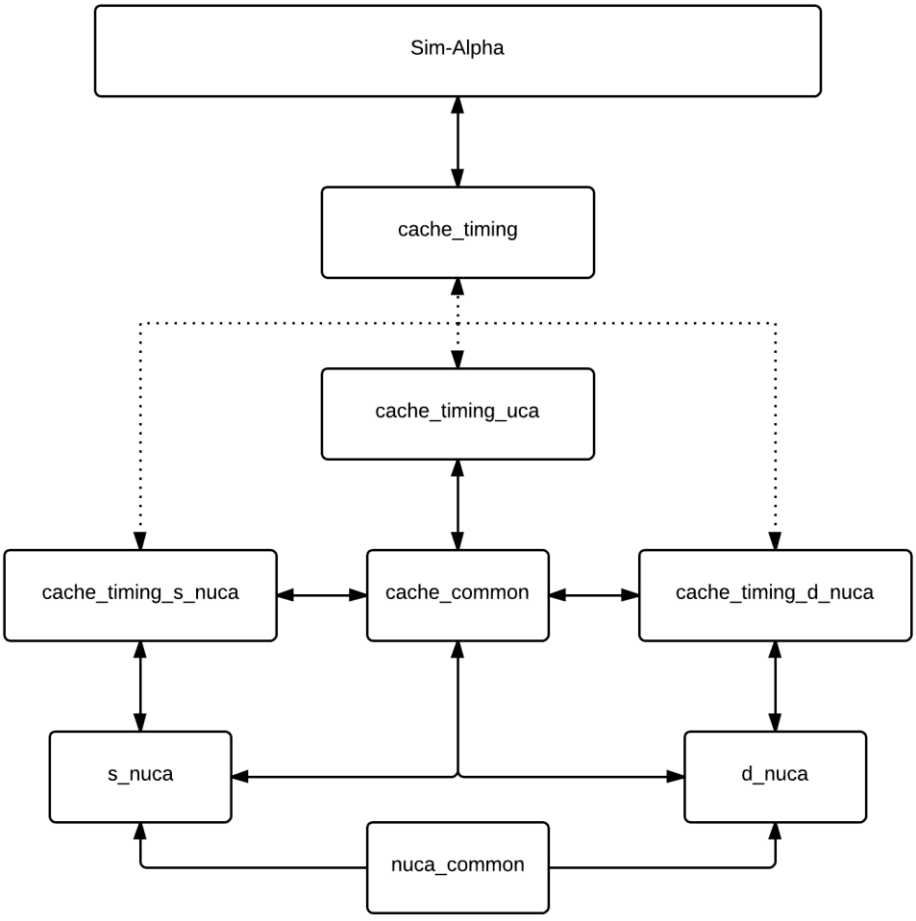


Figura 2 : Architettura della cache all'interno del simulatore sim-alpha

Di seguito sono descritti i vari moduli che compongono lo schema in figura e implementano la gerarchia di cache all'interno del simulatore. La descrizione è puramente logica.

- ***cache_common :*** contiene le definizioni delle strutture dati e delle funzioni comuni a tutte le topologia di cache
- ***cache_timing :*** modulo di interfaccia per le operazioni sulla cache per l'analisi temporale. Fornisce un interfaccia al simulatore che prescinde dalle implementazioni per le varie cache.
- ***cache_timing_s_nuca :*** Implementazione dell'interfaccia per l'analisi temporale , specifica per la cache s_nuca.
- ***cache_timing_d_nuca :*** Implementazione dell'interfaccia per l'analisi temporale , specifica per la cache d_nuca.
- ***cache_timing_uca :*** Implementazione dell'interfaccia per l'analisi temporale , specifica per la cache uca.
- ***nuca_common :*** struttura dati e funzioni comuni per i due tipi di nuca cache
- ***s_nuca :*** funzioni specifiche per l'accesso ai banchi s-nuca
- ***d_nuca :*** funzioni specifiche per l'accesso alle righe di cache d-nuca

In seguito verrà mostrato com'è stato implementato il meccanismo di scelta all'interno del simulatore e le opzioni per la selezione vera e propria.

1.2.2. Implementazione del meccanismo di scelta

L'implementazione del meccanismo di scelta deve ricalcare la struttura presentata nel precedente paragrafo , ma ,poiché il simulatore è scritto interamente in C , l'uso di tali concetti non risulta di immediata applicazione, in quanto non esiste nessun costrutto del linguaggio in grado di implementare in maniera diretta il concetto di derivazione. Per ovviare a questo inconveniente , l'implementazione effettiva è stata effettuata per mezzo dei puntatori a funzione. Come vedremo questa scelta implementativa facilita sia l'introduzione del meccanismo di scelta stesso , che l'inclusione di eventuali moduli aggiuntivi.

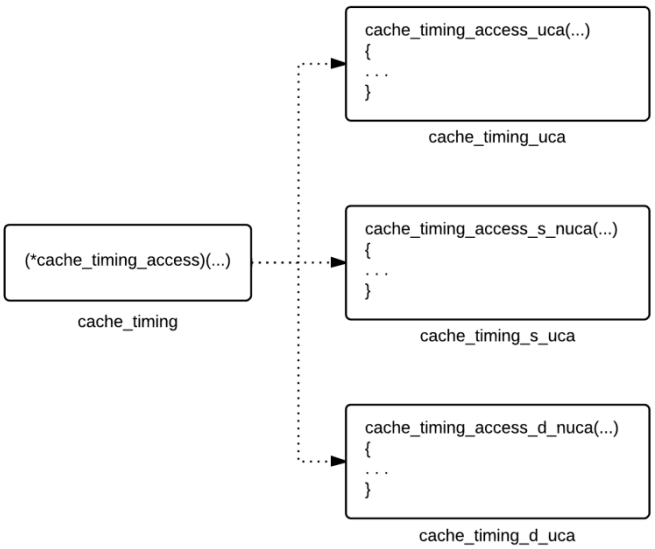
Procedendo con l'implementazione del sistema , per prima cosa occorre separare il moduli monolitico *cache_timing* ,in moduli specifici per ciascuna tipologia di cache implementata, i cui nomi sono assegnati seguendo lo schema riportato in tabella codice (1). All'interno di ciascun modulo software ,è contenuta la stessa serie di funzioni pubbliche, ed anche in questo caso il nome di ciascuna funzione porta al suo interno il nome della cache che implementa come riportato in tabella codice (2).

```
<nome_modulo>_<nome_cache>     (1)

<nome_funzione>_<nome_cache>   (2)
```

Codice 1 : Schema del nome delmodulo software

Il passo successivo consiste nella creazione di un modulo di interfacciamento con il sistema *cache_timing* al cui interno sono creati una serie di puntatori a funzione a cui verranno associate le funzioni presenti all'interno del vari moduli specifici. Tramite questi particolari puntatori il simulatore , in maniera trasparente e senza interventi ulteriori , potrà facilmente chiamare la chiamate per la gestione temporale della cache. Per limitare al minimo le modifiche all'interno del simulatore , il nome del modulo di interfacciamento è lo stesso del vecchio modulo monolitico che conteneva tutte le implementazioni insieme. In figura è riportato un esempio con la funzione *cache_timing_access*



In ultima istanza occorre creare le opzioni e le funzioni per l'associazione della cache selezionata. La scelta di quale modulo usare deve essere fatta in fase di configurazione, quando vengono letti i dati contenuti nei file passate all'avvio. In particolare tale configurazione deve risiedere all'interno del file di configurazione della cache globale , essendo una risorsa condivisa da tutti i processori. Per questo motivo deve essere inserita una nuova opzione , che permetta di specificare quale modulo usare , all'interno del file di configurazione della cache. La sistassi della nuova opzione è riportata in tabella codice

```
-nuca:type              S_NUCA|D_NUCA|UCA
```

Codice 2 : : Opzione per la selezione della tipologia di cache

Una volta registrata tale opzione occorrerrà procedere con l'assegnazione effettiva dei puntatori , che viene fatta tramite un appostita funzione del nuovo modulo *cache_timing* chiamata *associate_cache()*.

1.2.3. Implementazione del supporto alle cache S-NUCA

In precedenza , Il simulatore in uso (sim-alpha_d-nuca_cmp) supportava solo le tipologie D-NUCA e UCA, in modalità single e multicore . Per aggiungere il supporto alla tipologia di memorie S-NUCA ci siamo basati sull'implementazione di tale cache , codificata in un altro simulatore ,solo single core, e abbiamo provveduto ad estendere tale implementazione per supportare la

multiprogrammazione. Tale Implementazione consiste essenzialmente in una modifica dell'implementazione basilare delle D-NUCA a cui sono state apportate le seguenti modifiche:

- Rimozione del processo di migrazione
- Inserimento in tutta la cache e non solo nell'ultima via (come accade per le D-NUCA)

Il porting è stato effettuato utlizzando lo scheda in figura 1.

1.3. Estensione delle politiche di risparmio energetico per le cache S-NUCA

La riduzione del consumo di potenza è un obiettivo sempre più importante per le moderne CPU, sia che siano applicate per dispositivi mobili , sia in sistemi ad alte prestazioni. Oltre alla potenza dinamica , dovuta alle commutazioni interne, con l'aumento delle dimensioni delle cache nei moderni processori commerciali il consumo di potenza statica ,assume sempre più rilievo. Date le grandi dimensioni , le cache sono i candidati ideali per operare una riduzione della potenza statica. Per questo sono state sviluppate molte tecniche di risparmio energetico per ridurre il consumo di potenza.

All'interno del simulatore erano già state implementate le principali tecniche di risparmio di energia statica per le cache L2 sulle cache D-NUCA. In particolare erano state implementate le seguenti tecniche:

- Drowsy
- Decay
- Way Adattable

Con l'introduzione del supporto alle cache S-NUCA , queste tecniche sono state estese , qualora fosse possibile, anche a questa nuova tipologia di cache. In particolare è stato implementato il supporto per le sole tecniche Decay e Le Drowsy. La politica Way Adattable non è stata estesa perché risulta priva di senso applicata alle cache S-NUCA. Questo metodo infatti, utilizzando il meccanismo della migrazione , proprio della D-NUCA ,consiste nello spengere le ultime vie della cache che non sono utilizzata per contenere dati, in quanto sono migrati nelle vie ad accesso più veloce.

1.3.1. La tecnica Drowsy

La tecnica drowsy consiste nel porre le linee di cache in uno stato low-power , ridicendo opportunamente il voltaggio operazionale di standby. Prima di ogni accesso a tali cache line, il loro voltaggio di funzionamento deve essere ristabilito al valore nominale , per non distruggere il dato contenuto, incorrendo però in una latenza ulteriore. Per limitare tale latenza aggiuntiva ,una volta che una cella è stata acceduta , si può provvedere a tenerla all'alimentazione nominale. Periodicamente , ogni intervallo di riconfigurazione W (espresso in cicli di clock), un semplice

algoritmo stabilisce quali linee porre nel cosidetto “drowsy mode”. L’algoritmo segue le seguenti politiche per scegliere se le linee devono essere spente

- Simple : tutti i blocchi vengono messi in modalità Drowsy ogni W cicli
- Noaccess : se la linea non è stata acceduta negli W cicli di clock predecenti , allora viene messe in modalità Drowsy

È possibile inoltre scegliere se porre sia i campi tag e data nello stato drowsy , oppure solo i campi data , lasciando i tag sempre attivi

Ne consegue che combinando le modalità precedenti , si ottengono 4 modalità di funzionamento

- Simple ,tag e data drowsy
- Simple , solo data drowsy
- Noaccess tag e data drowsy
- Noaccess solo data drowsy

Tramite l’esplorazione di queste modalità si può arrivare a soluzioni ottimali che garantiscono un risparmio del consumo statico di potenza , senza inficiare pesantemente sulle prestazioni

L’applicazione di tale tecnica all’interno del simulatore era stata già implementata per le cache D-NUCA , e UCA , quindi si è provveduto solo ad estenderla per le cache S-NUCA appena implementate. In seguito è illustrata brevemente tele implementazione. Per ogni linea di cache si mantengono alcune informazioni , come lo stato in cui si trova, l’ultimo accesso. Gli stati in cui una linea si può trovare sono :

- ACTIVE : linea attiva
- DROWSY : linea in modalità drowsy , a basso consumo statico
- WAKING_UP : la linea era spenta, ma è stata acceduta , per cui occorre riportare la tensione di alimentazione al livello nominale; tale linea sarà attiva dopo un certo tempo (up transition time)
- GOING TO SLEEP : la linea era attiva ma l’algoritmo ha deciso di spengerla; si troverà nello stato DROWSY dopo un periodo di tempo (down transition time)
- POWERED DOWN : la linea è senza alimentazione (valido per l’esecuzione in contemporanea con il metodo Way adattable per le cache D-NUCA).

Durante l’esecuzione della simulazione , l’esecuzione di azioni sulle linee di cache (read/write miss , read/write hit , writeback ecc) comporta una modifica e di conseguenza un accensione della linea. Periodicamente, utilizzando il sistema di gestione degli eventi del simulatore, viene eseguita una routine che , sulla base dello stato in cui si trova la linea , decide se porla o meno nello modalità drowsy.

Per estendere questo algoritmo anche alle cache S-NUCA sono stati eseguite le seguenti modifiche:

- *Introduzione dei ritardi dovuti all’attivazione di una linea per le cache S-NUCA*

Questo obiettivo è stato ottenuto creando una nuova funzione per la gestione dell'accesso in cache , modificando il modulo ***bus*** , che si occupa degli accessi al bus. Per usare tale modifica si è reso necessario modificare anche il modulo ***s_nuca*** per garantire che la nuova routine fosse eseguita e per la gestione dei parametri necessari alla sua esecuzione.

- *Inseriment della gestione delle azioni durante il funzionamento anche per la S-NUCA*

  Questo obiettivo è stato ottenuto modificando il modulo ***s_nuca*** , aggiungendo la gestione delle azioni che sono fondamentali per il corretto funzionamento dell'algoritmo. In particolare , in ciascuna delle funzioni che gestisce la creazione di pacchetti di routing della cache , sono stati aggiunti i parametri relativi alla azione in corso.

- *Inserimento del codice di controllo per l'attivazione dell'algoritmo*

  È stato aggiunto il codice per l'abilitazione dell'algoritmo , modificando il modulo ***simulate***

Una volta effettuate le seguenti modifiche , il codice S-NUCA è in grado di interfacciarsi correttamente con l'implementazione precedente dell'algoritmo

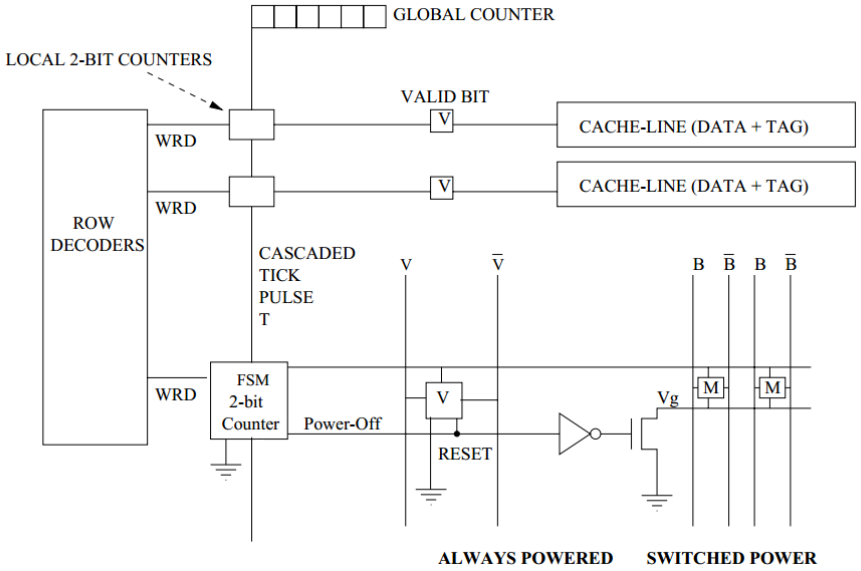
1.3.2. La tecnica Decay

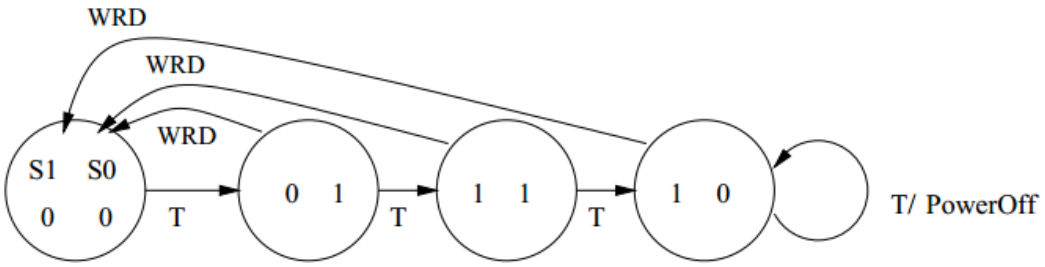
Il concetto su cui si basa la tecnica di risparmio energetico Decay consiste nel fatto che di solito , durante il ciclo di vita di una linea di cache , intercorre un periodo di tempo relativamente lungo , tra il momento in cui è stata acceduta per l'ultima volta e la completa cancellazione. L'idea quindi è di andare ad intervenire su questo periodo "morto", spengendola completamente senza causare ulteriori cache miss. Lo spegnimento della linea avviene tramite l'utilizzo della tecnica GatedVDD , che consiste nel chiudere l'alimentazione ad una certa linea di cache , distruggendo così il dato all'interno . Una semplice politica per determinare se la linea è stata acceduta per l'ultima volta nel corso della sua vita consiste nel misurare l'intervallo di tempo che intercorre tra l'ultimo accesso noto e analizzare il tempo di inatività un intervallo di tempo prestabilito in fase di costruzione. Occorre quindi aggiungere un registro per ciascuna delle linee della cache per permettere di salvare il tempo il tempo di inattività della singola linea. Questo approccio però risulta impossibile da applicare perché troppo oneroso, considerando l'imgombro di tale logica di controllo sul die. Per questo, si utilizza un sistema di conteggio gerarchico , in cui un contatore golobale comanda contatori più piccoli , posti su ciascuna della linee di cache, come mostrato in figura .

Come si può notare , il contatore globale comanda in cascata i singolo contatori delle linee di cache, ciascuno dei quali implementa una macchina a stati finiti , che definisce lo stato della singola linea di cache , come riportato in figura

Quando il contatore globale ha raggiunto il limite di tempo prefissato , genera l'impulso che comanda l'avanzamento della macchina a stati (T). Gli accessi alla linea (WRD) resettano lo stato della linea riportandolo alla stato iniziale. Se la linea raggiunge l'ultimo stato , questa è spenta. In

questo modo si raggiunge un compromesso tra occupazione di spazio della logica di controllo ed efficienza dell'algoritmo.





Per migliorare l'efficienza dell'algoritmo , sono state presentate varie soluzioni , come ad esempio il predittore IATAC che consente una di attuare predizioni più ampie , basandosi sui pattern di accesso per effettuare una predizione sulla globalità della cache e non sulla singola linea , come avviene per l'algoritmo base . Questo è motivato essenzialmente dal fatto che la tecnica di gatedVdd è un tecnica distruttiva per le informazioni esistenti , quindi , per evitare accessi onerosi in memoria off chip , occorre affinare il meccanismo di scelta della linee da disattivare. Per aumentare le prestazioni , e ridurre gli spegnimenti erronei, IATAC osserva il tempo di inter-access per ogni linea, classifica queste informazioni a seconda del numero di accessi alla linea, e fa previsioni sulla base del storia globale.

Gli algoritmi prcedentemente descrtti , erano già implementati all'interno del simulatore ,come per l'algoritmo Drowsy, e attivabili solo per le cache D-NUCA e UCA. Il lavoro svolto consiste estendere il supporto anche per le cache S-NUCA. I passi seguito sono i seguenti:

- *Introduzione del meccanismo di aggiornamento dello stato della linea per l'algoritmo base,durante un accesso ad essa*

  È stato modificato il modulo ***bus*** del codice del simulatore, introducendo una funzione specifica di accesso alla cache,in modo da eseguire per l'aggiornamento della macchina a stati finiti della singola linea e per la modifica della statistiche della simulazione

- *Introduzione del meccanismo di aggiornamento dello stato della linea per l'algoritmo IATAC ,durante un accesso ad essa*

  È stato modificato il modulo **cache_timing_access_s_nuca** per l'inserimento delle chiamate a funzione per l'aggiornamento delle informazioni gestite dal predittore IATAC , per il corretto funzionamento dell'algoritmo.

1.3.3. Metodologia di verifica della correttezza

Per verificare il corretto funzionamento delle modifiche illustrate in precedenza , oltre che per via analitica , verificando funzionalmente la correttezza di ciascuna modifica , si è ricorso all'analisi sperimentale dei risultati. Abbiamo poi confrontato i risultati ottenuti con l'applicazione dei metodi di risparmio energetico sulle cache S-NUCA con i risultati analoghi ottenuti per le D-NUCA. Questo è motivato dal fatto che entrambi gli algoritmi agiscono esclusivamente sulle linee di cache per cui su cache della stessa dimensione , con lo stesso numero di linee di cache , per la stessa simulazione , entrambe le tecnologie utilizzeranno lo stesso numero di linee di cache , per l'esecuzione del programma simulato. Le differenze tra le varie tecnologie riguarderanno il mapping all'interno dei banchi ,e il metodo di ricerca dei dati all'interno dei banchi. Seguendo questa idea è stato costruito un meccanismo di debug per tenere traccia dello stato delle linee di cache alla fine di ogni finestra temporale . In particolare abbiamo provveduto a tener traccia del numero di linee che si trovano in un determinato stato.

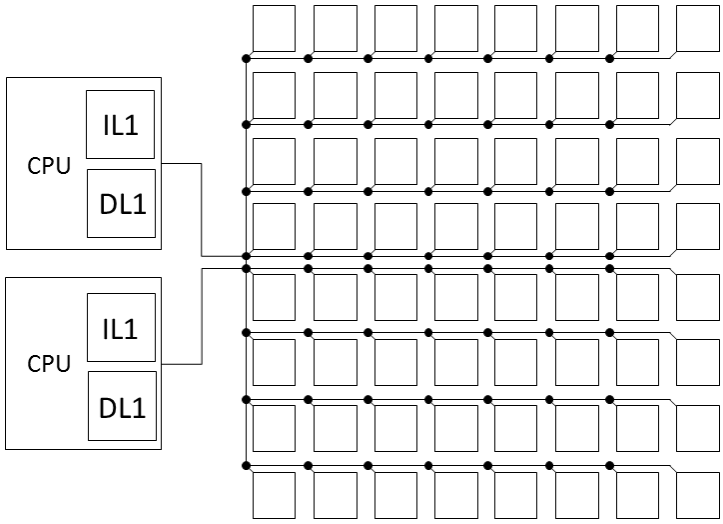
1.4. Il calcolo della potenza istantanea

Un'altra importante feature inserita all'interno del simulatore , consente di estrarre le informazioni per poter calcolare la potenza istantanea. In particolare è stato implementato un sistema che permette di collezionare i dati relativi al consumo di potenza, ad intervalli regolari (specificati in cicli di clock), in modo da poter vedere come si evolve il consumo di potenza del processore durante una specifica simulazione. Il sistema , è stato concepito per testare l'impatto l'evoluzione dell'consumo energetico per cache D-NUCA , in presenza di algoritmi di risparmio energetico basati sullo spegnimento delle vie,come way halting. Per poter effettuare correttamente il calcolo occorre estrarre i seguenti dati dalle simulazioni in tabella.Per collezionare tali dati, si è reso necessario modificare ulteriormente il simulatore per calcolare alcuni dati che non erano presenti o valutare i dati presenti e ricavarli da essi. Successivamente inserire un nuovo modulo software per l'estrazione di tali dati e , in ultima istanza l'introduzione di nuove opzioni per l'attivazione di tale funzionalità.

Potenza Statica :	Numero di Banchi accesi/spenti
	Numero di Switch accesi/spenti
Potenza Dinamica :	Accessi in cache
	Trasmissioni sui Link/Switch
	Potenza per accensione ulteriore di vie
	Accessi in lettura/scrittura alla SDRAM

1.4.1. Scelta dei parametri

Per estrarre i dati significativi per il calcolo del controllo di potenza , siamo partiti dall'architettura di cache mostrata in figura per calcolare i vari parametri.



Andiamo adesso ad analizzare come sono stati calcolati i vari parametri:

- Numero di banchi accesi /spenti

  Il numero dei banchi attivi è sempre associato al numero di banchi presenti , qualora non si applicato un meccanismo di spegnimento delle vie come ad esempio l'algoritmo di risparmio energeto Way Halting. Qual ora un algoritmo di questo tipo sia in funzione , in numero di banchi variano rispetto al numero di vie attive seguendo le seguenti formule

$$N_{banchi\ attivi} = N_{vie\ attive} \cdot N_{righe\ nuca}$$

$$N_{banchi\ disattivati} = (N_{colonne\ nuca} - N_{vie\ attive}) \cdot N_{righe\ nuca}$$

  Il numero di vie attualmente attive è un parametro salvato all'interno del simulatore e facilemente reperibile.

- Numero di Switch accessi / spenti

L’architettura della cache in figura ci mostra che ,per ciascuna riga il numero degli switch presenti nel simulatore è uguale al numero dei banchi meno uno. Se non vi sono attive politiche di risparmio energetico , il numero degli switch attivi per riga ,è effettivamente quello calcolato in precedenza , ma se sono presenti politiche di risparmio energetico basate sullo spegnimento di vie come way hailting il numero di switch attivi varia seguento la formula

$$N_{switch\ attivi} = (N_{vie\ attive} - 1) \cdot N_{righe\ nuca}$$

$$N_{switch\ attivi} = (N_{colonne\ nuca} - N_{vie\ attive} - 1) \cdot N_{righe\ nuca}$$

- Accessi in lettura/scrittura in cache

  Il simulatore mette già disposizione le statistiche necessarie per valutare gli accessi alla cache

- Trasmissioni sui Link/Switch

  Il simulatore mette già a adisposizione le statistiche necessarie per valutare il numero di accessi ai link , operando la distinzione tra accesso a link verticali o link orizzontali. Per quanto riguarda gli accessi agli switch , il simulatore non da nessun dato diretto , ma , osservando la l’architettura della cache si nota che ogni trasmissione sui link comporta necessariamente una trasmissione sugli switch.

- Accensione di ulteriori via

  Qualora è si applicato un meccanismo di spegnimento delle vie come ad esempio l’algoritmo di risparmio energeto Way Halting, è possibile che la cache abbia una riconfigurazionene durante l’esecuzione della simulazione, aumentando o diminuendo il numero di linee attive. Nel caso in cui venga aggiunta una linea di cache , oltre ad aumentare il consumo di energia statica , vi è un aumento temporaneo della energia dinamica,consumata per effettuare la procedura di attivazione. Per tener conto di questo consumo addizionale , si tengono traccia di tutte le riconfigurazioni della cache utilizzando la seguente formula.

  $$= X_{global} - X_{old}$$

- Accessi in lettura/scrittura in SDRAM

  Il simulatore mette a disposizione un parametro unico raggruppa gli accessi in lettura e in scrittura in un'unica statistica. Per avere un indicazione più precisa della potenza dissipata , presumendo che le potenze necessarie per compiere queste due operazioni sia diverse, si è operato a separare le statistiche riguardanti le due operazioni.

Una volta scelte le statistiche passiamo a creare il modulo software la gestione di tali dati.

1.4.2. Implementazione

Per l'abilitazione e la configurazione del processo di campionamento sono stati create delle opzioni apposite ,illustrate nella tabella codice

```
-power:enabled          1/0
-power:window_size      <n_cicli_clock>
-power:log_file     <file_di_salvataggio>
```

Codice 3 : Parametri di configurazione per il calcolo del consumo di potenza

Tramite il parametro *window_size* si può settare il valore della finestra di campionamento su cui effettuare la raccolta dei dati. Per facilitare il reperimento dei risultati ottenuti, si può specificare un file su cui salvare i dati . Un formato utile per il salvataggio dei file è il formato cvs (Comma Separated Vector) leggibile direttamente dai fogli elettronici più comuni.

Passando all'implementazione effettiva della funzionalità , È stato creato un nuovo modulo chiamato ***power*** che si occupa sia di gestire l'abilitazione del servizio , che di collezionare e salvare i vari risultati creati durante le simulazione. All'avvio della simulazione , se la funzionalità è stata abilitata , si creno le strutture dati , si inizializzano , e si programma , tramite il gestore degli eventi implementato all'interno del simulatore , la riescecuzione periodica della funzione di campionamento ,con frequenza specificata dal parametro *window_size*. Per ogni finestra temporale , viene scritta una riga sul file specificato dall'opzione *log_file.* I dati relativi alla potenza statica , sono calcolabili direttamente vengono calcolati in base alle formule descritte in precedenza. Invece i dati relativi alla potenza dinamica nella finestra corrente ,sono calcolati tramite sottrazione del valore attuale del parametro globale con il valore assunto dallo stesso , nella precendete finestra temporale,come mostrato dalla formula.

$$window\ param_t = param_t - param_{t-1}$$

VALUTAZIONE E CONFRONTO TRA LE VARIE TECNOLOGIE

2.1. Analisi della potenza istantanea per le cache D-NUCA

In questo paragrafo ci occuperemo di analizzare l'evoluzione del consumo di potenza per le cache D-NUCA ,in cui è applicato l'algoritmo di risparmio energetico Way Halting. L'algoritmo sfrutta la non uniforme distribuzione dei dati nelle cache D-NUCA per adattare il numero di banchi attivi alle esigenze dell'applicazione in corso ,attivando o disattivando dinamicamente le vie che compongono la cache. Lo studio presentato è stato realizzato confrontando i consumi di diverse implementazioni dell'algoritmo ,al fine di valutarne le caratteristiche.

2.1.1. Architettura di riferimento

I nuclei simulati sono basati sulla microarchitettura Alpha 21264, in esecuzione auna frequenza di 4 GHz, in tendenza con le linee microprocessore di fascia alta in un processo a 32 nm. In Tabella 1 sono riportate le caratteristiche della cache scelta per il confronto

	D-NUCA
Dimensione:	8 MB
Dimensione Linea	64 B
N. banchi	128
Dimensione Banco	64 KB
N. righe	16
N. colonne	8
Associatività	Direct Mapped
Tecnologia	32 nm

Tabella 1 : caratteristiche della Cache D-NUCA

Le configurazioni scelte per il confronto sono :

- plain : cache D-NUCA standard in cui non è applicato nessun algoritmo di risparmio energetico
- WA_No_Thr_No_Osc : cache D-nuca a cui è applicato la versione dell'dell'algoritmo way halting senza le soglie e senza oscillazioni
- WA_AvgThr: cache D-nuca a cui è applicato la versione dell'dell'algoritmo way halting con le soglie calcolate in maniera euristica.

Per ricavare i parametri relativi alla potenza è stato utilizzato lo strumento CACTI 5.1 ,modellando un banco di 64KB alla tecnologia di 32nm.Per effettuare l'analisi abbiamo scelto un tempo ti

campionamento pari a 1 milione di cicli di clock , valore deciso su base sperimentale andando ad analizzare i periodi medi in cui avviene una riconfigurazione della cache.

2.1.2. Analisi Single Core

Per effettuare l'analisi per abbiamo simulato le diverse configurazione della cache elencate in seguito con i benckmark della siute SPEC2000 elencate in Tabella 2

Benchmark	Suite	FFWD	RUN
Gcc	SPECINT2000	2.367B	300M
Parser	SPECINT2000	3.709B	200M
Perlbmk	SPECINT2000	5.0B	200M
Twolf	SPECINT2000	511M	200M
Applu	SPECFP2000	267M	650M
Art	SPECFP2000	2.2B	200M
Equake	SPECFP2000	4.459M	200M
Galgel	SPECFP2000	4.0B	200M
Mesa	SPECFP2000	570M	200M
mgrid	SPECFP2000	550M	1.06B

Tabella 2 : benchmark utilizzati per l'analisi

Andiamo ad analizzare per prima cosa i dati della potenza istantanea per cache D-NUCA per ciascuna delle simulazioni coinvolte

2.1.2.1. Gcc

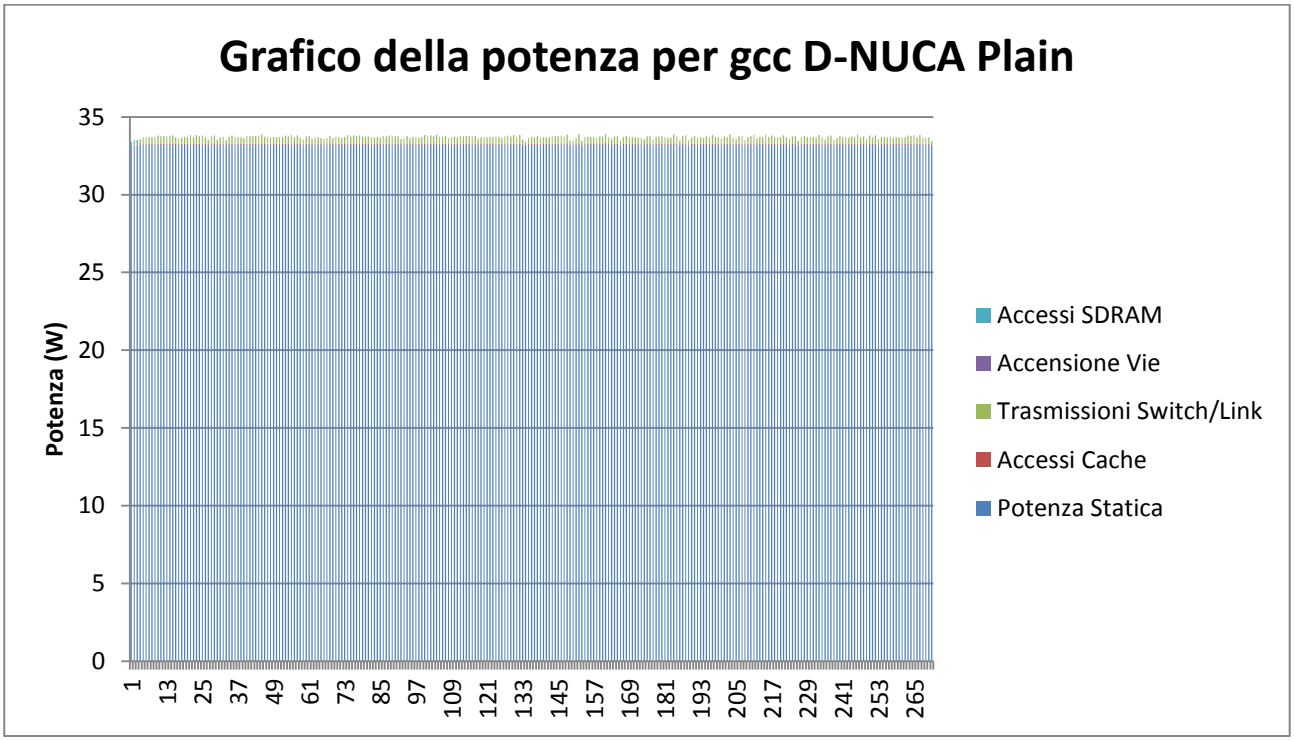


Grafico 1 : Grafico della potenza per gcc D-NUCA Plain

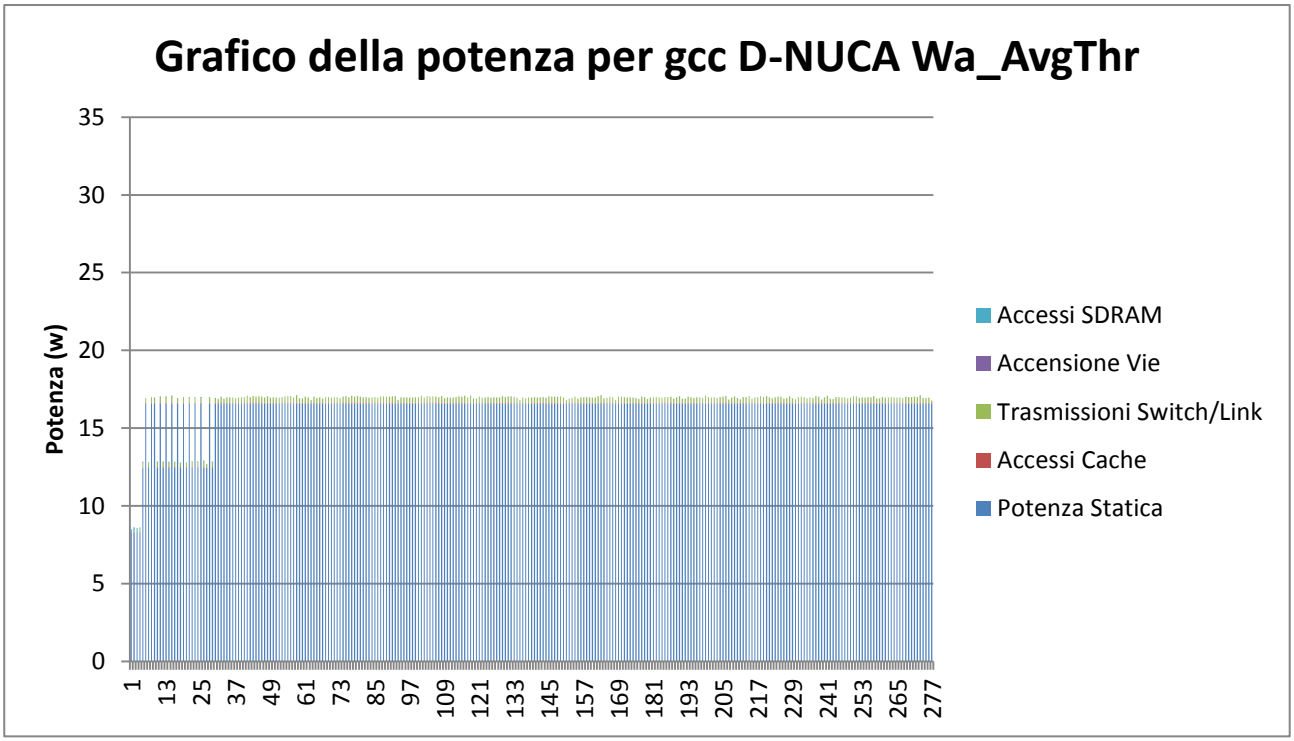


Grafico 2 : Grafico potenza per gcc Wa_AvgThr

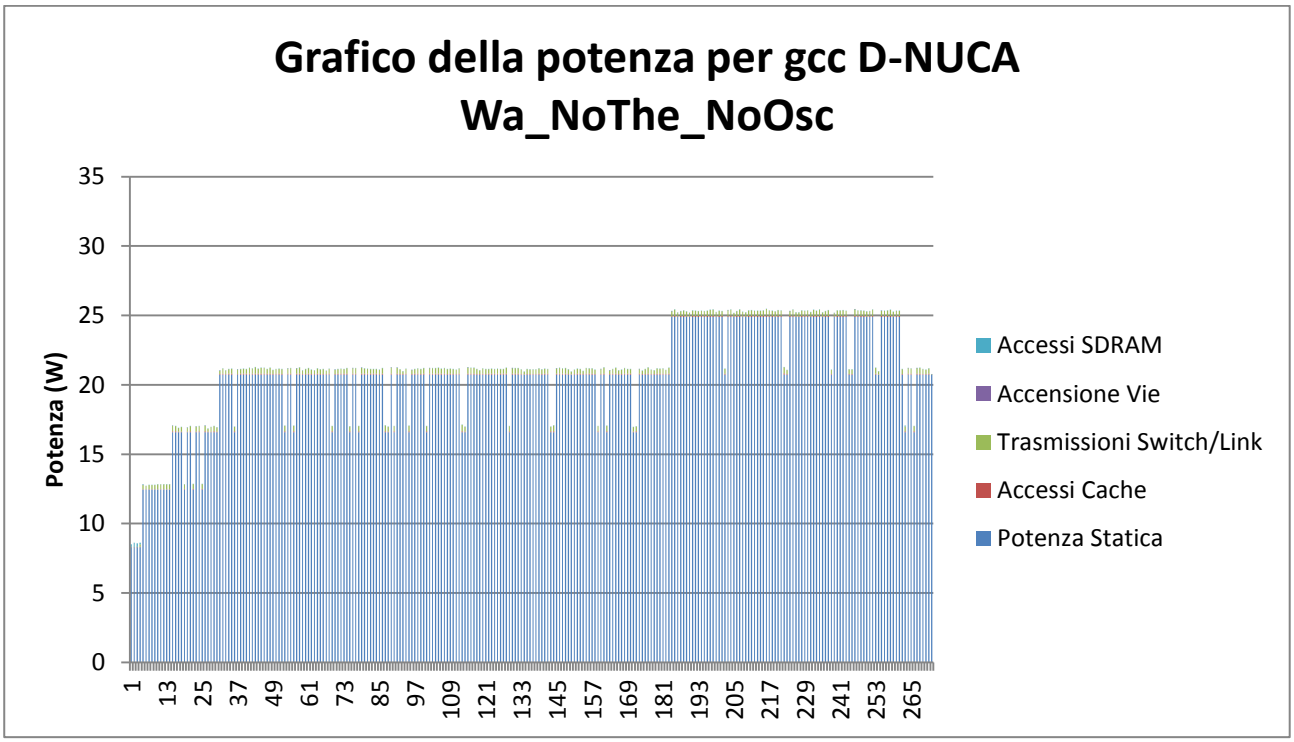


Grafico 3 : Grafico della potenza per gcc D-NUCA Wa_NoThr_NoOsc

Cache	Valore
Plain	33,87326 W
Wa_AvgThr	17,10338 W
Wa_NoThr_NoOsc	25,45913 W

Tabella 3 : Peak Power per gcc

2.1.2.2. Parser

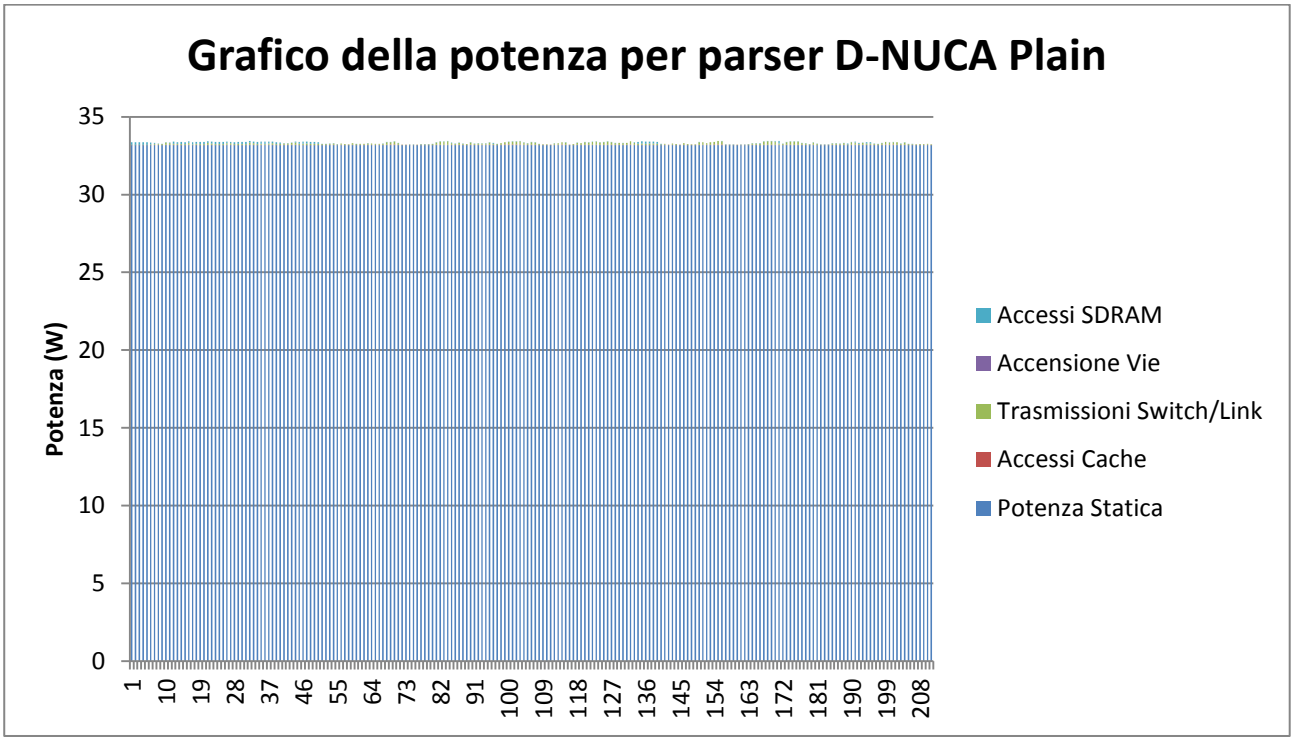


Grafico 4 : Grafico della potenza per parser D-NUCA Plain

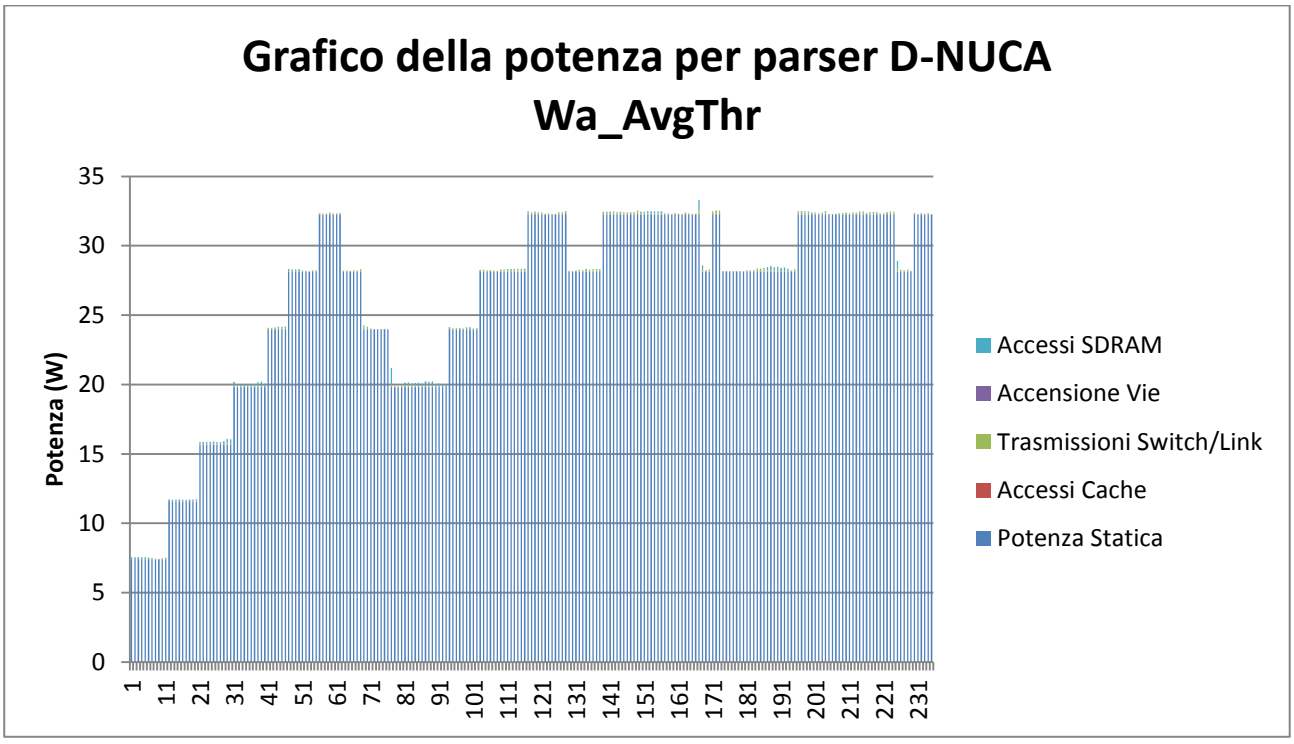
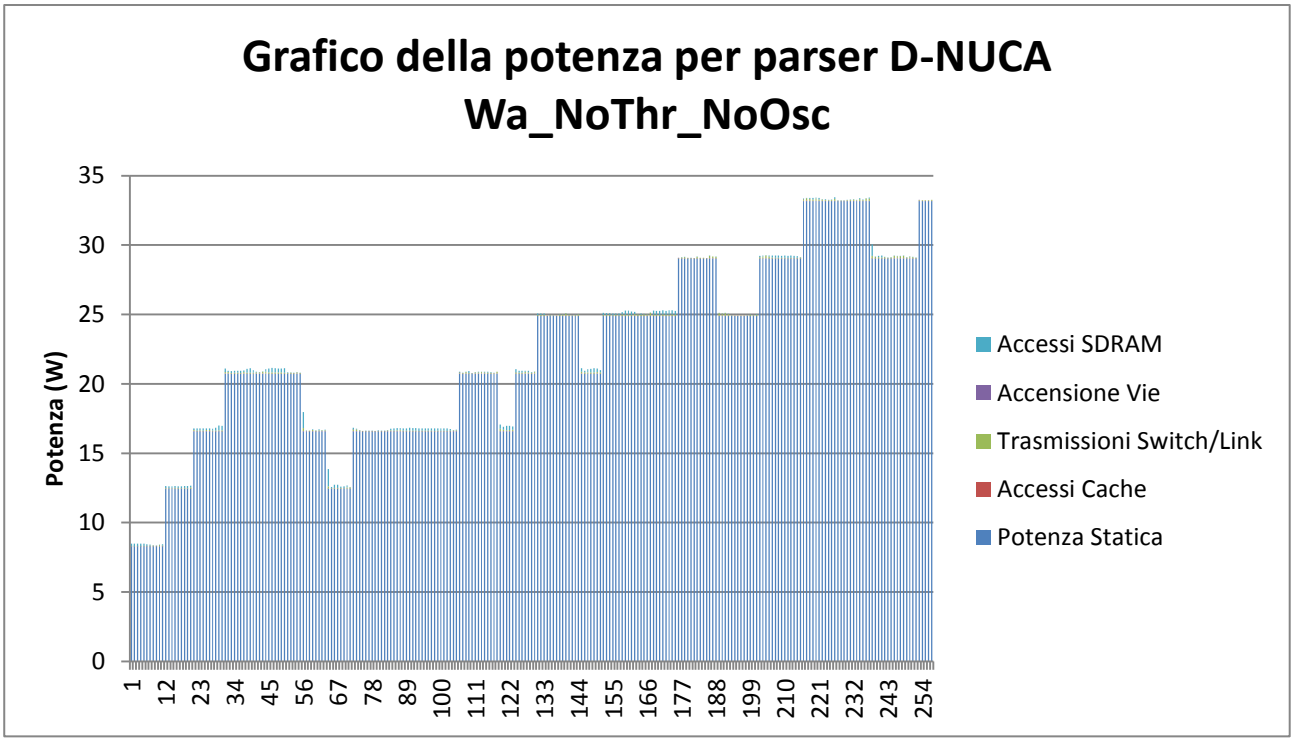


Grafico 5 : Grafico della potenza di parser D-NUCA Wa_AvgThr



Cache	Valore
Plain	33,45018 W
Wa_AvgThr	33,30295 W
Wa_NoThr_NoOsc	33,44969 W

Tabella 4 : Peak Power per parser

2.1.2.3. Perlbmk

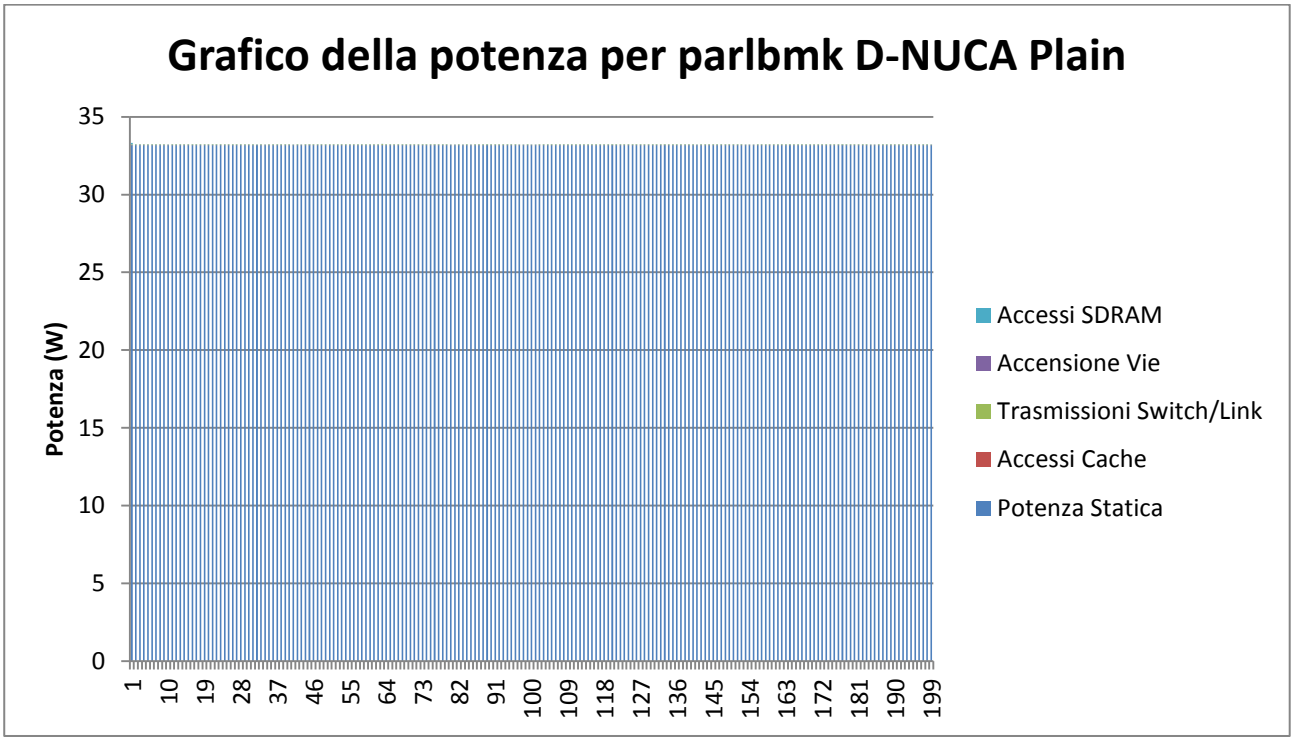


Tabella 5 : Grafico della potenza per perlbmk D-NUCA Plain

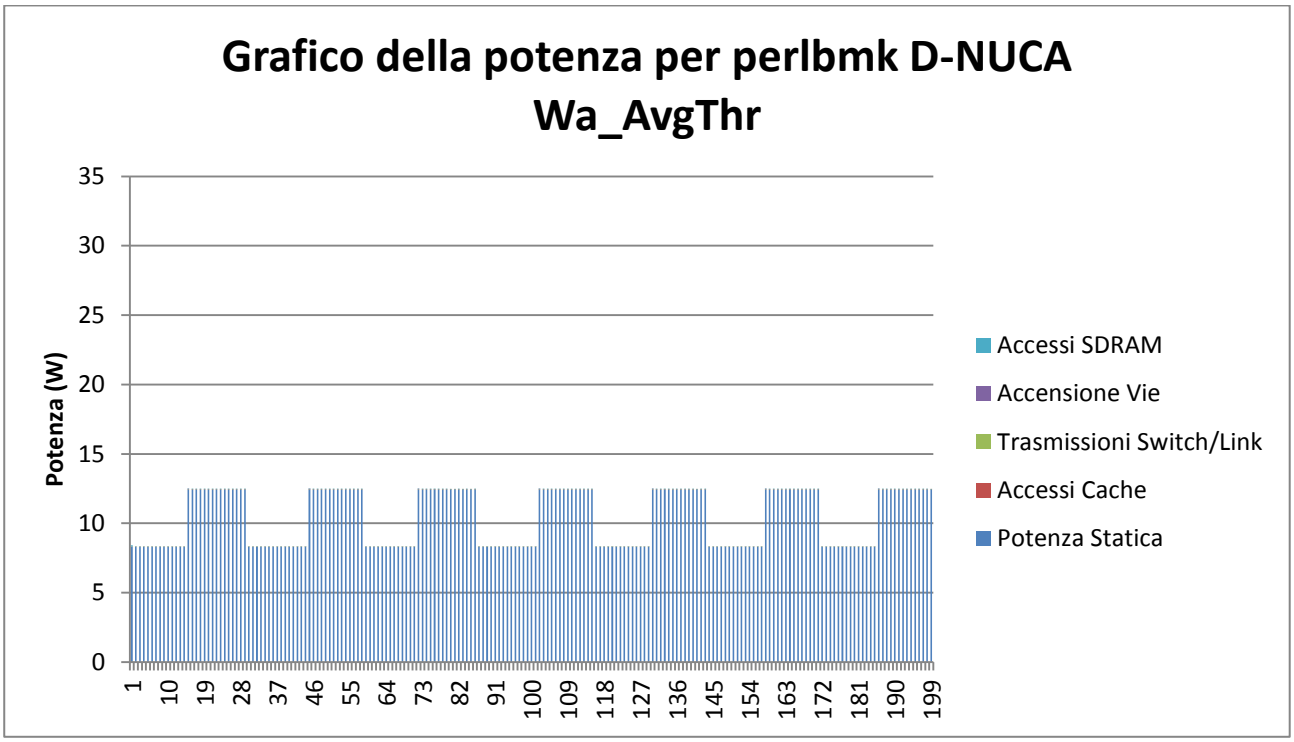
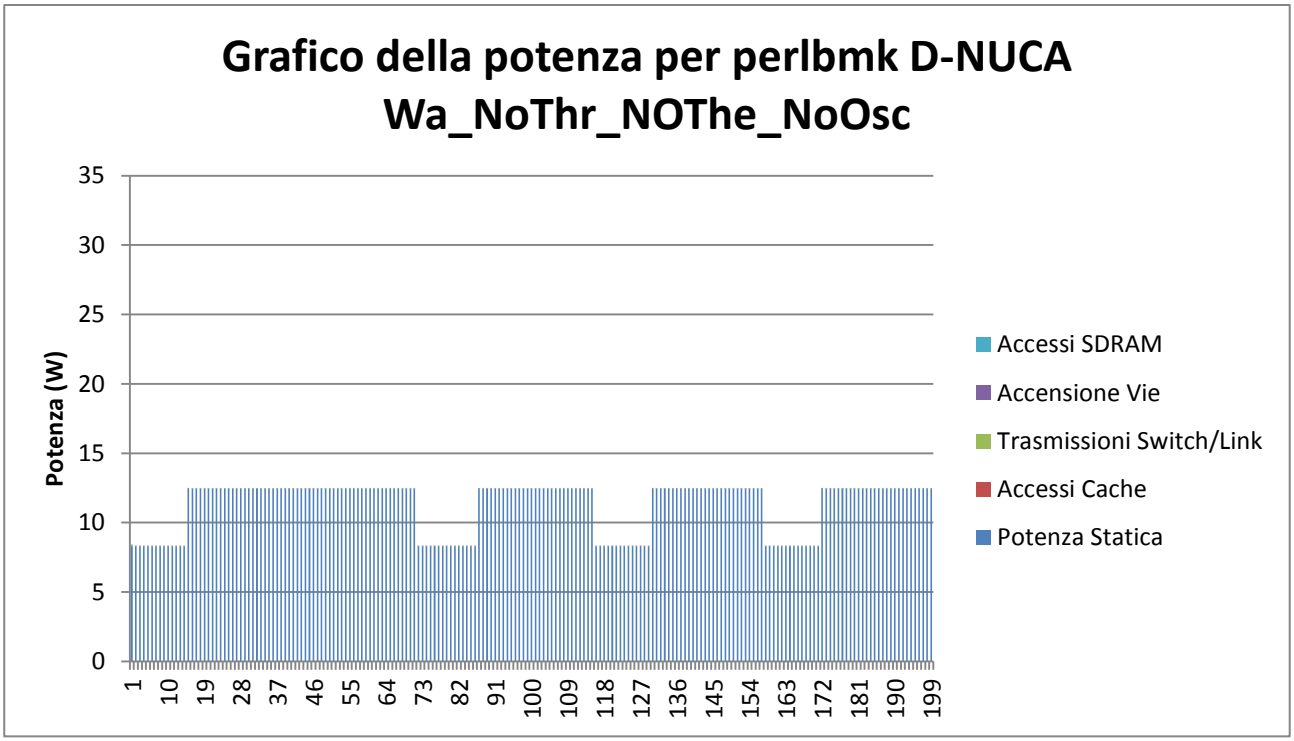


Tabella 6 : Grafico della potenza per perlbmk D-NUCA Wa_AvgThr



Cache	Valore
Plain	33,31845 W
Wa_AvgThr	12,47845 W
Wa_NoThr_NoOsc	12,48283 W

Tabella 7 : Peak Power per perlbmk

2.1.2.4. Twolf

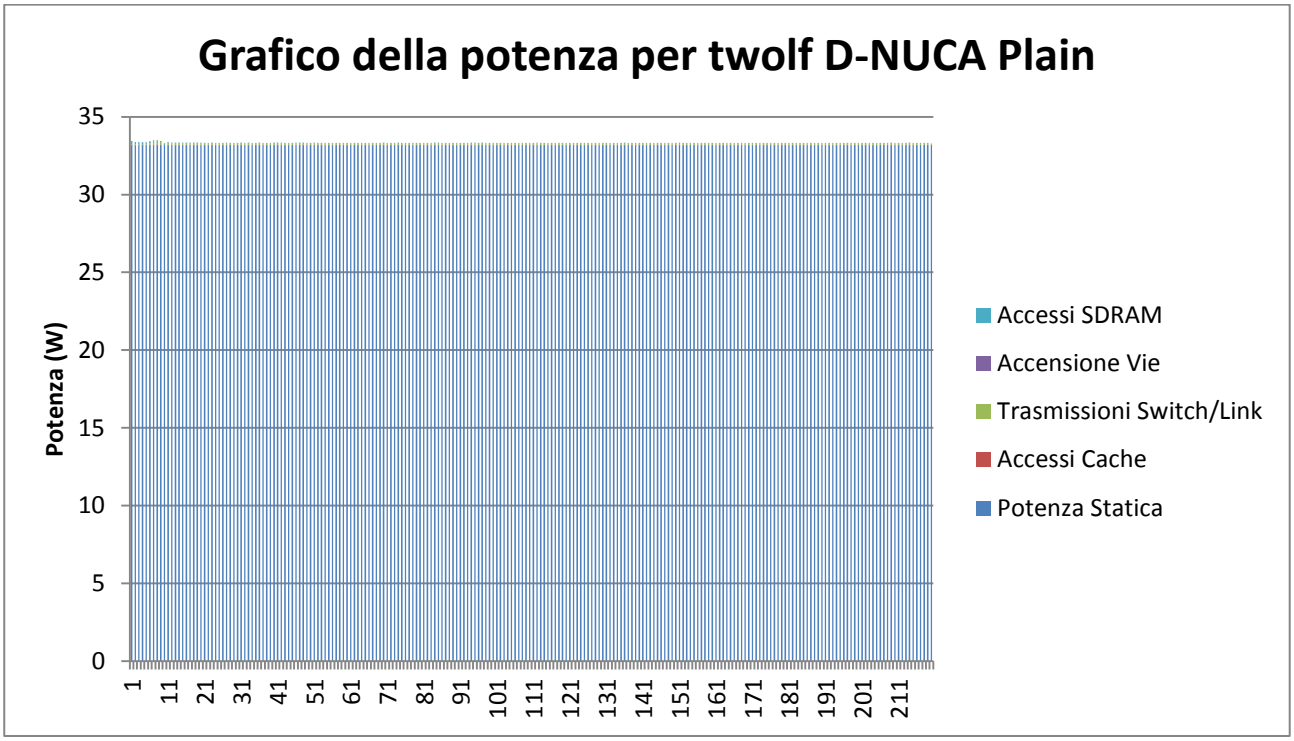


Tabella 8 : Grafico della potenza per twolf D-NUCA Plain

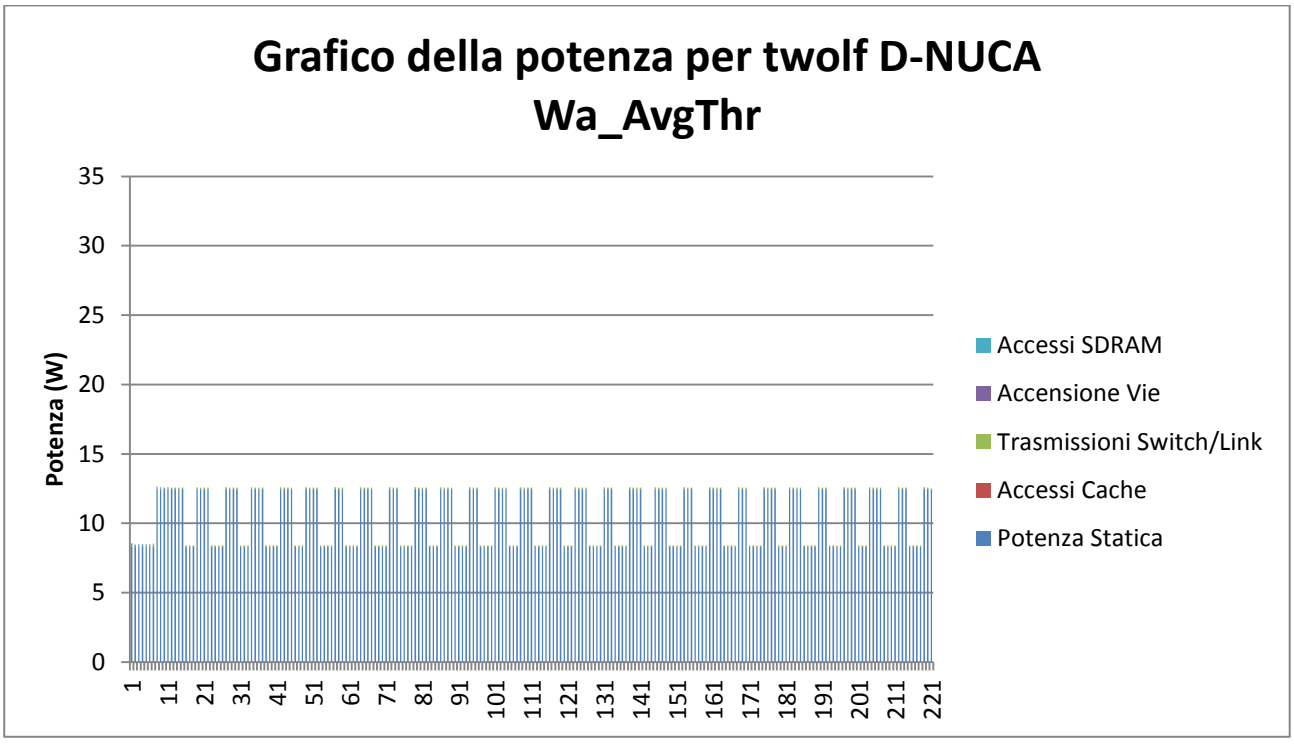


Tabella 9 : Grafico della potenza per twolf D-NUCA Wa_AvgThr

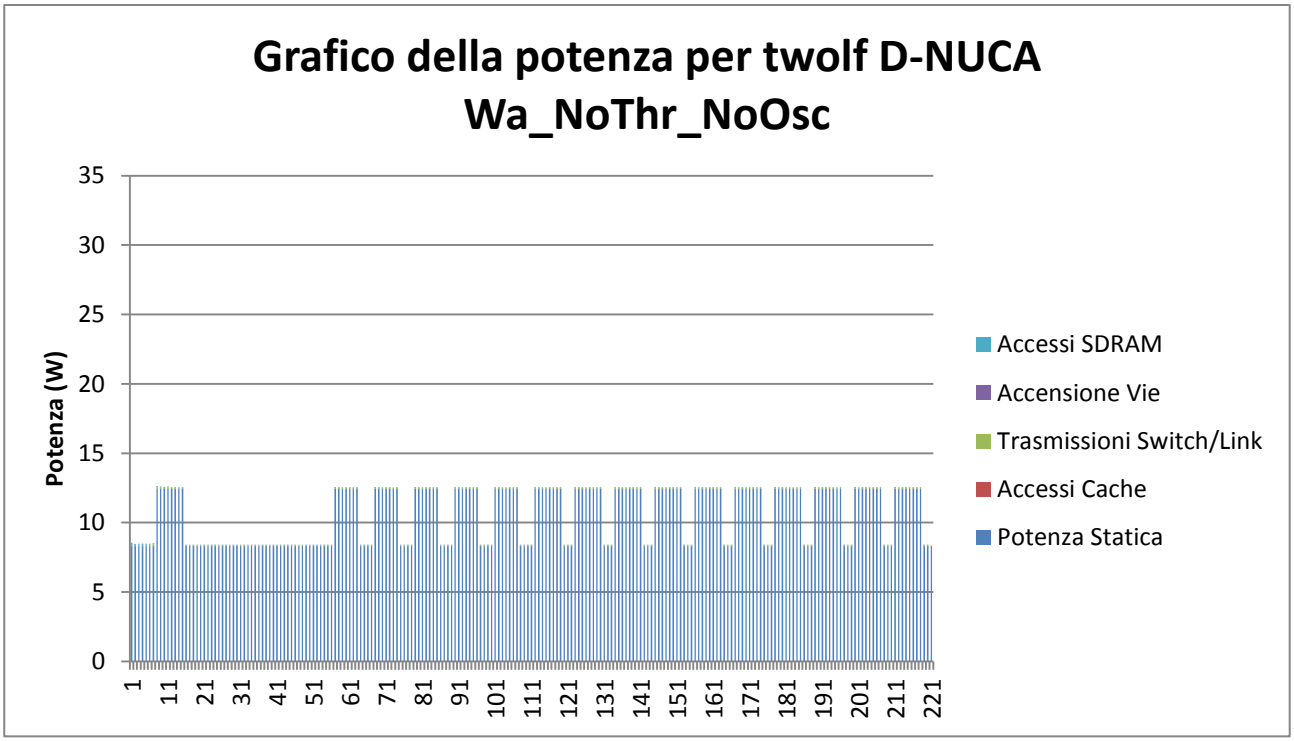


Tabella 10 : Grafico della potenza per twolf D-NUCA Wa_NoThr_NoOsc

Cache	Valore
Plain	33,49781 W
Wa_AvgThr	12,64635 W
Wa_NoThr_NoOsc	12,48283 W

Tabella 11 : Peak Power per twolf

2.1.2.5. Art

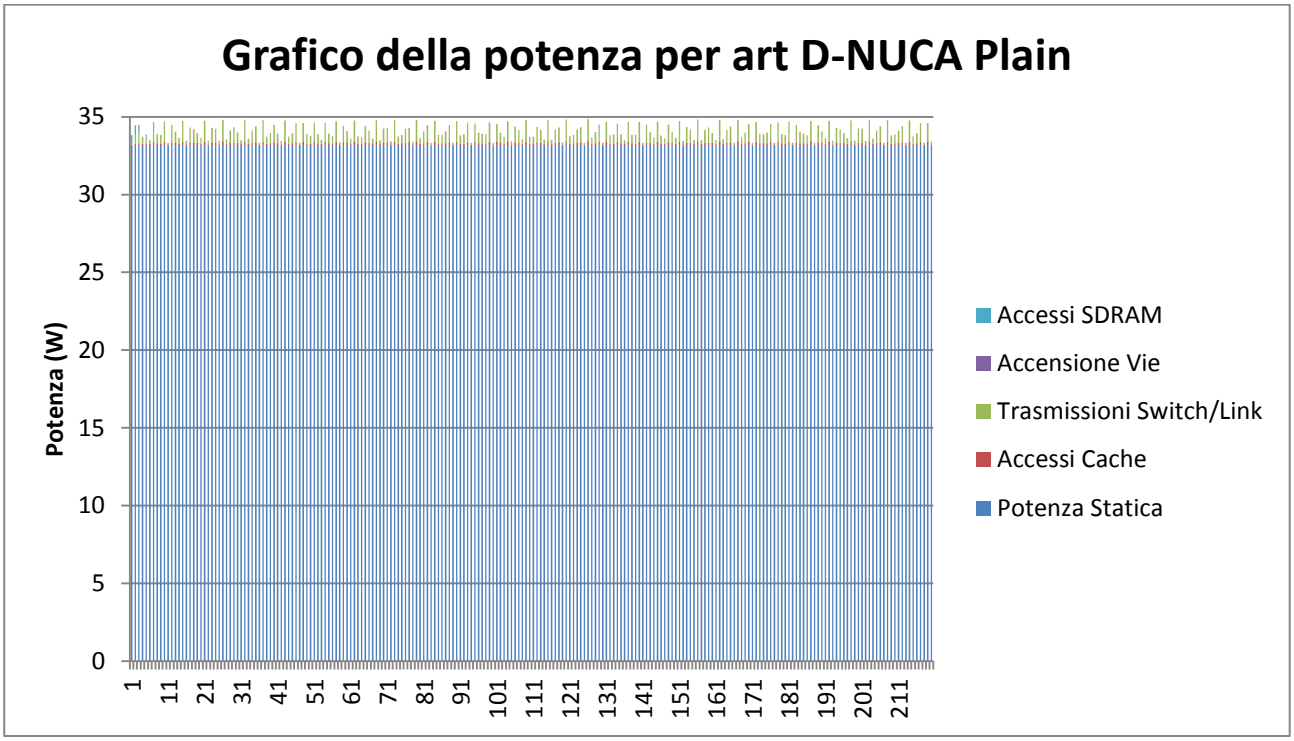


Tabella 12 : Grafico della potenza per art D-NUCA Plain

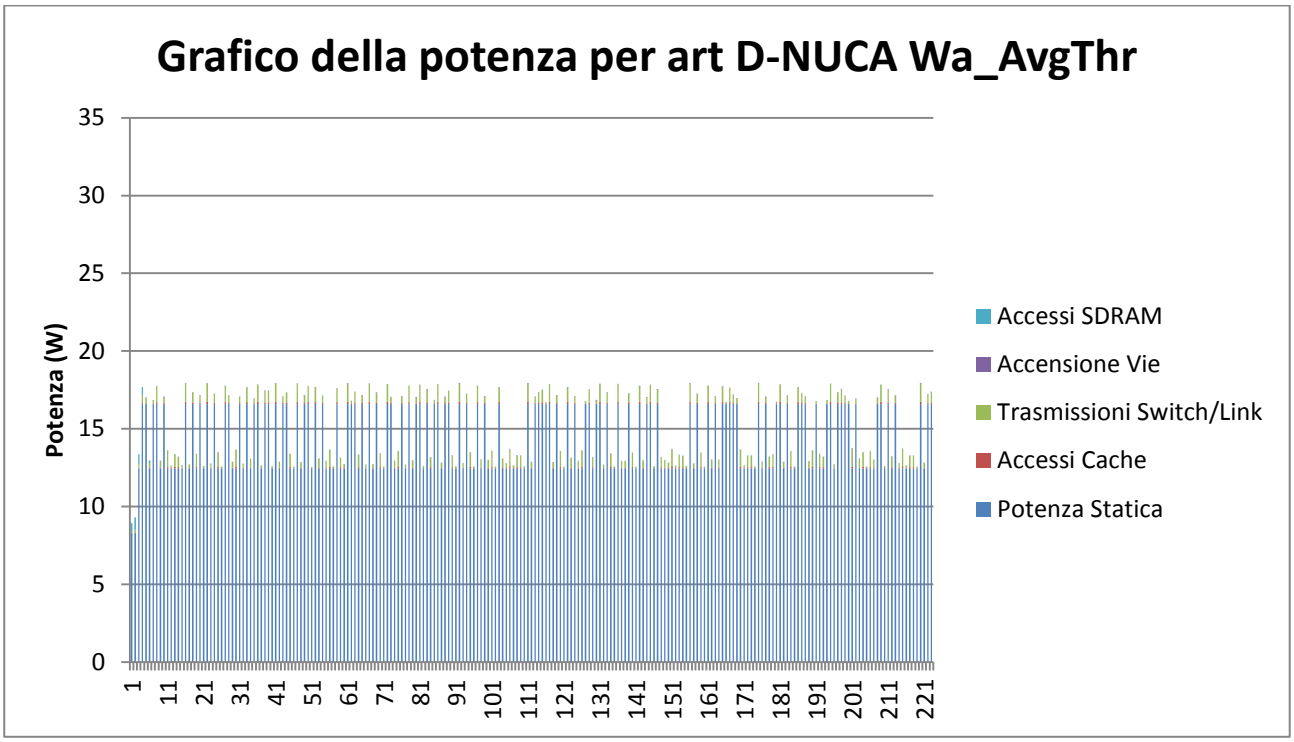


Grafico 6 : Grafico della potenza di art D-NUCA Wa_AvgThr

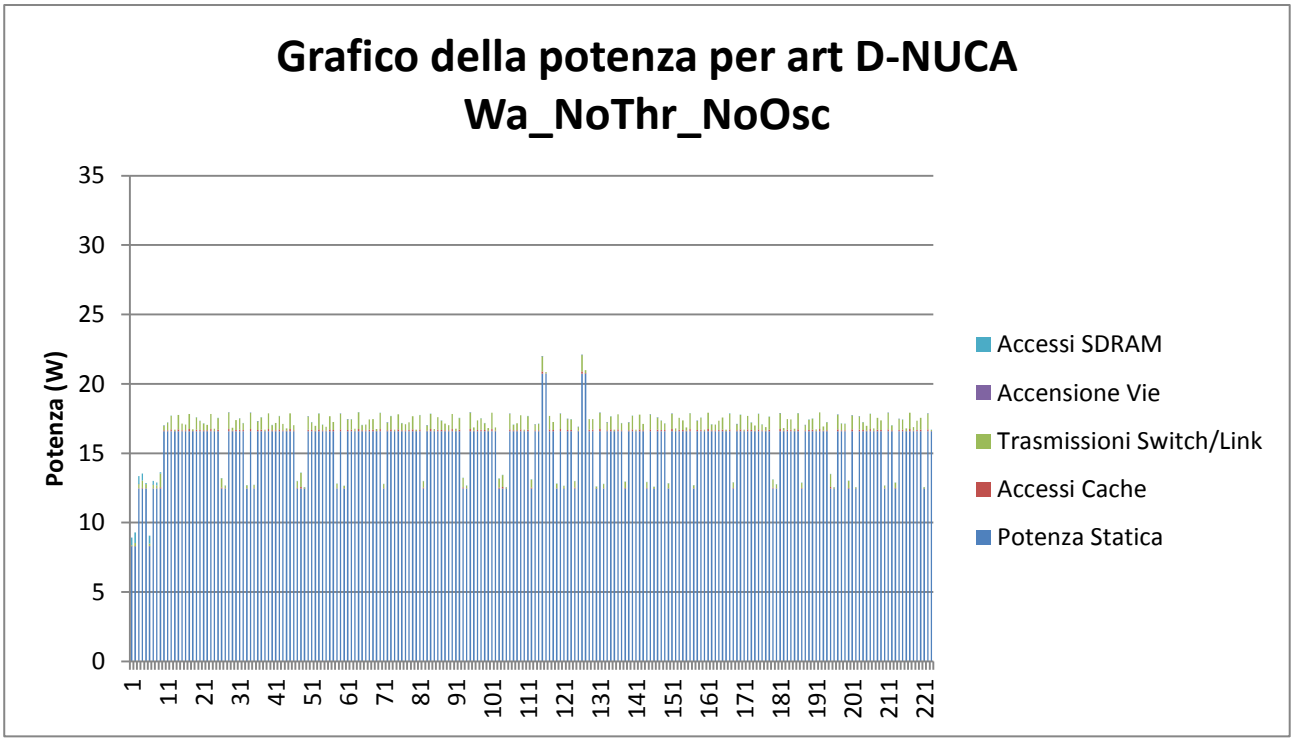


Grafico 7 : Grafico della potenza di art D-NUCA Wa_NoThr_NoOsc

Cache	Valore
Plain	34,83862 W
Wa_AvgThr	17,93984 W
Wa_NoThr_NoOsc	22,07429 W

Tabella 13 : Peak Power per art

2.1.2.6. Equake

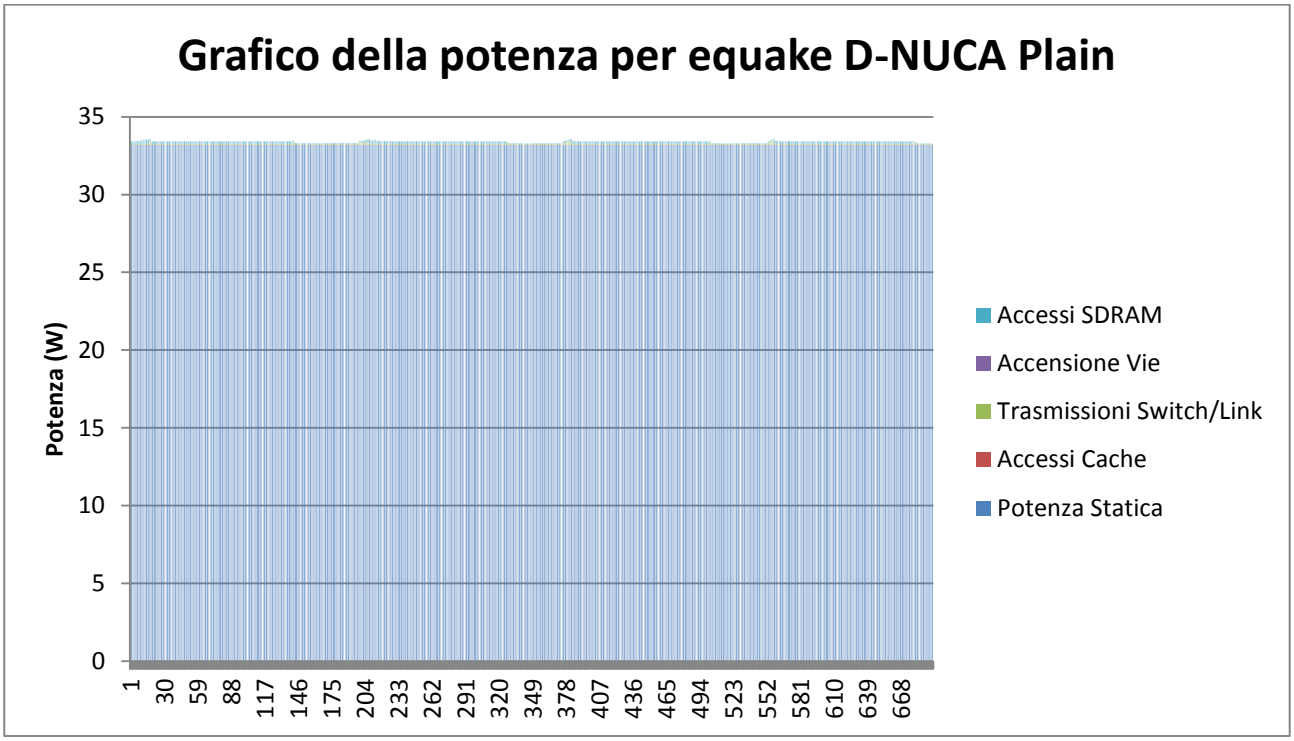


Grafico 8 : Grafico della potenza di equake D-NUCA Plain

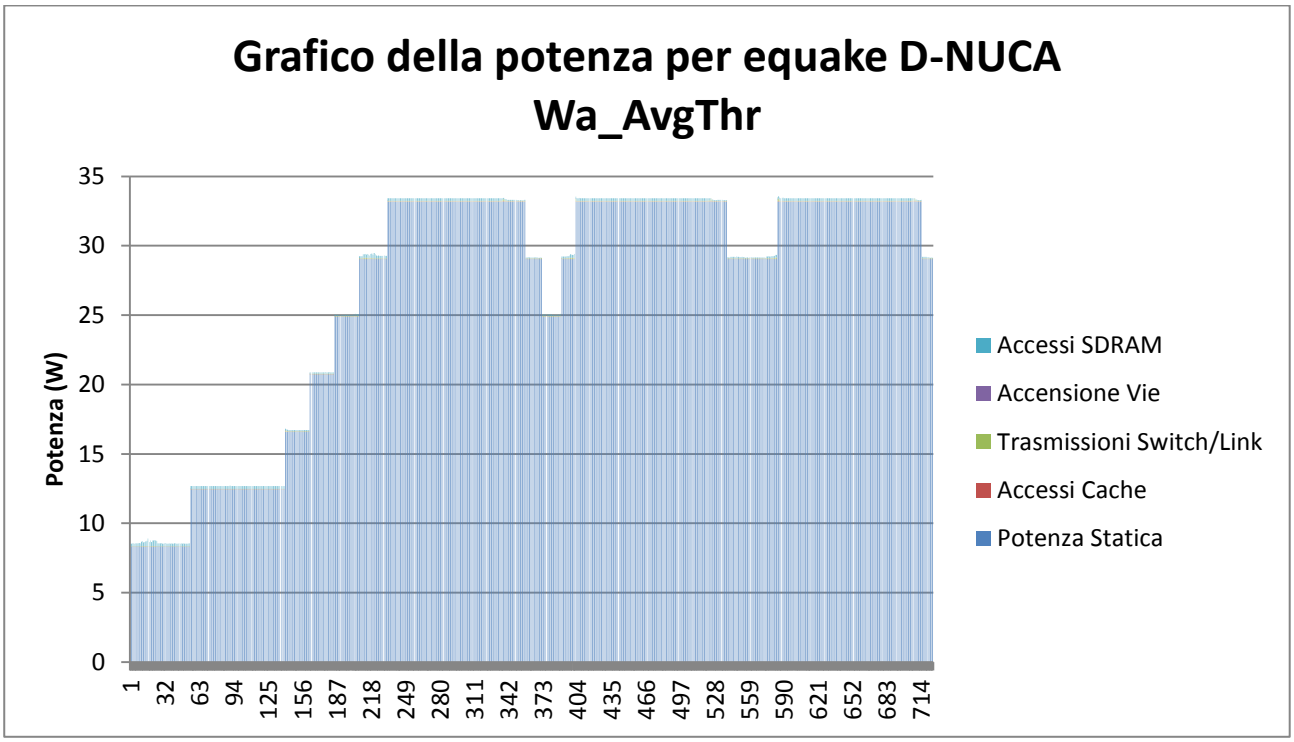


Grafico 9 : Grafico della potenza di equake D-NUCA Wa_AvgThr

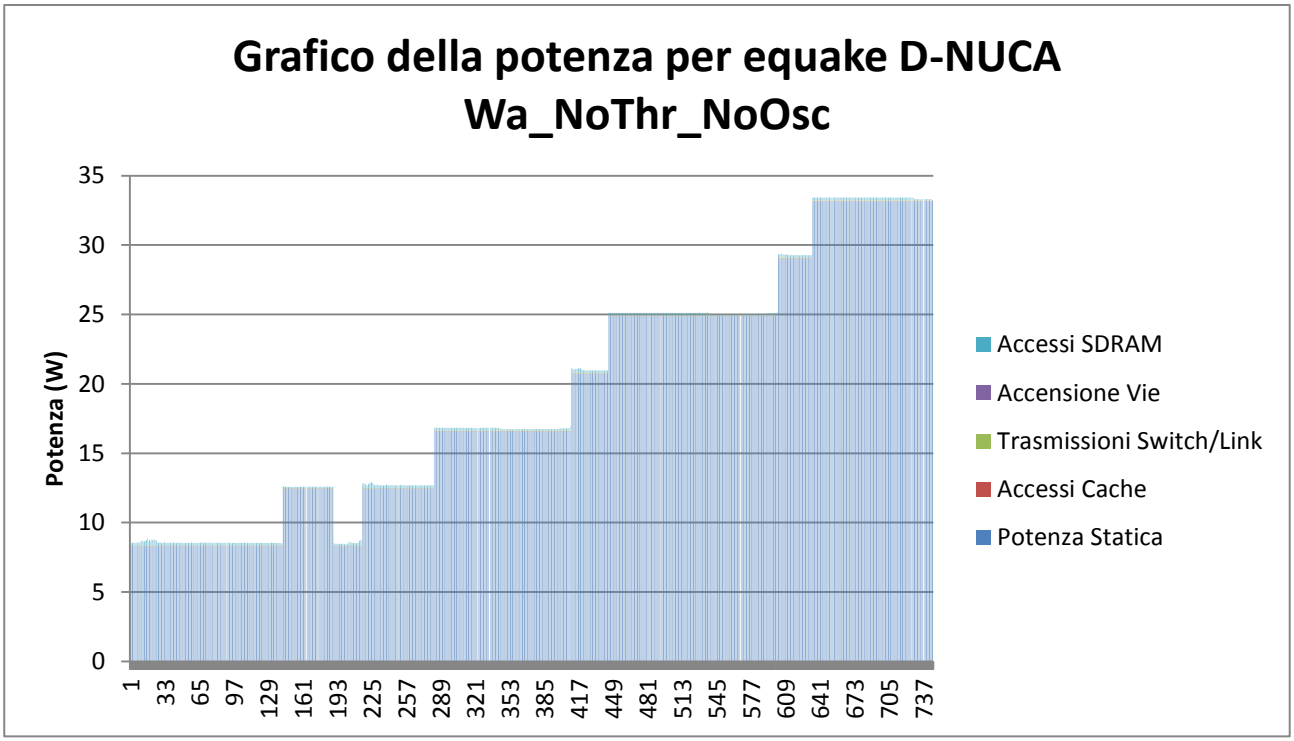


Grafico 10 : Grafico della potenza di equake D-NUCA Wa_NoThr_NoOsc

Cache	Valore
Plain	33,61588 W
Wa_AvgThr	33,556 W
Wa_NoThr_NoOsc	33,44087 W

Tabella 14 : Peak Power per equake

2.1.2.7. Galgel

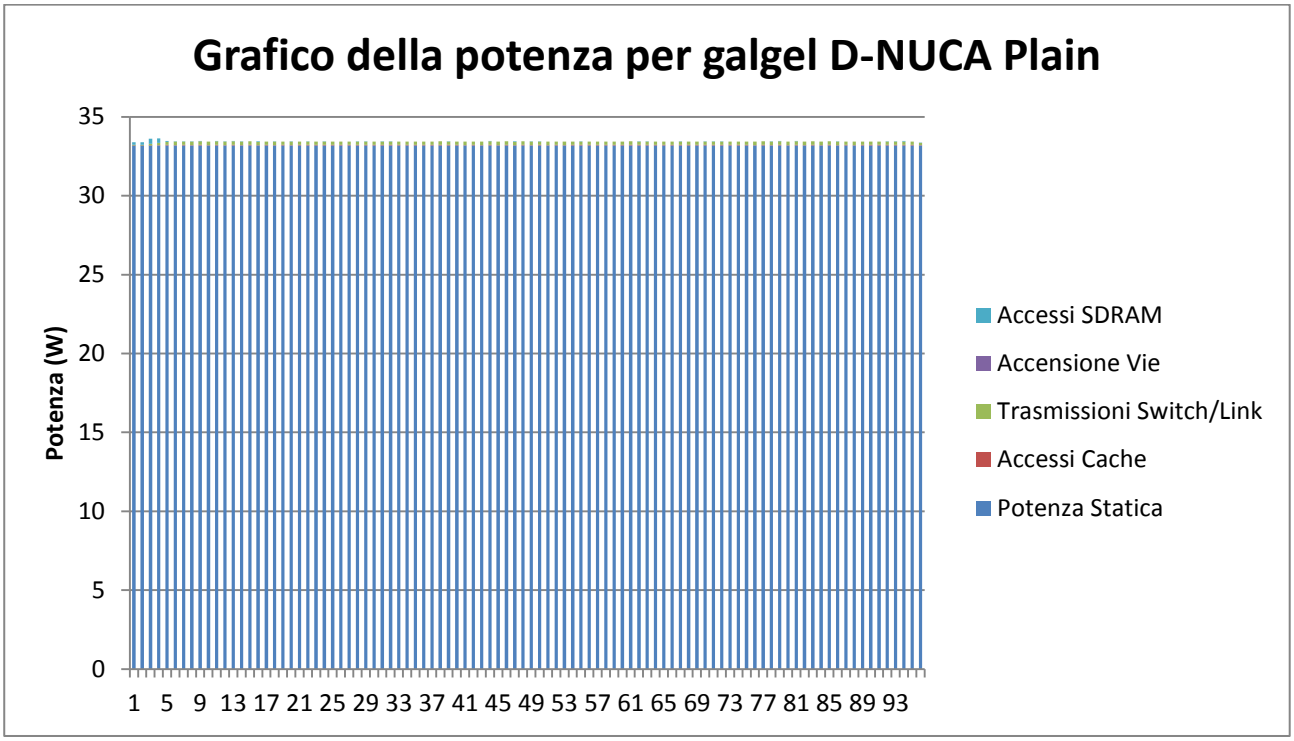


Grafico 11 : Grafico della potenza di galgel D-NUCA Plain

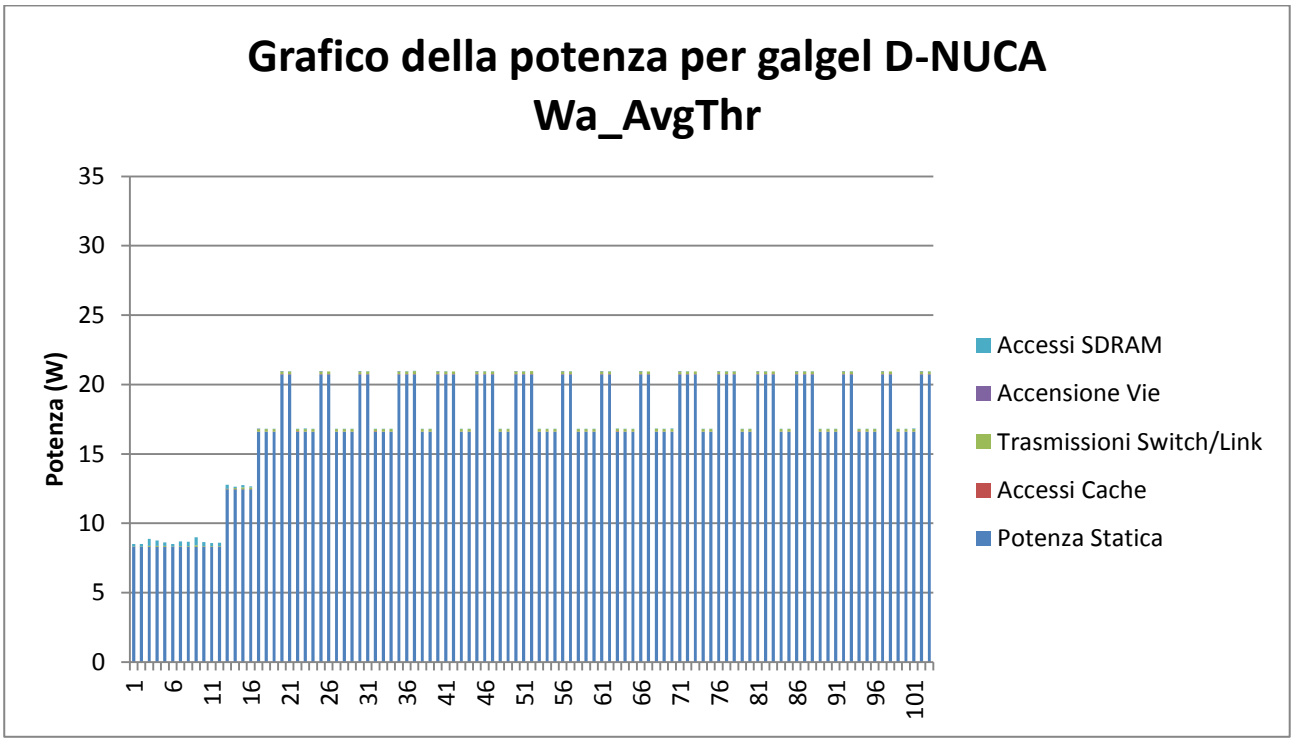


Grafico 12 : Grafico della potenza di galgel D-NUCA Wa_AvgThr

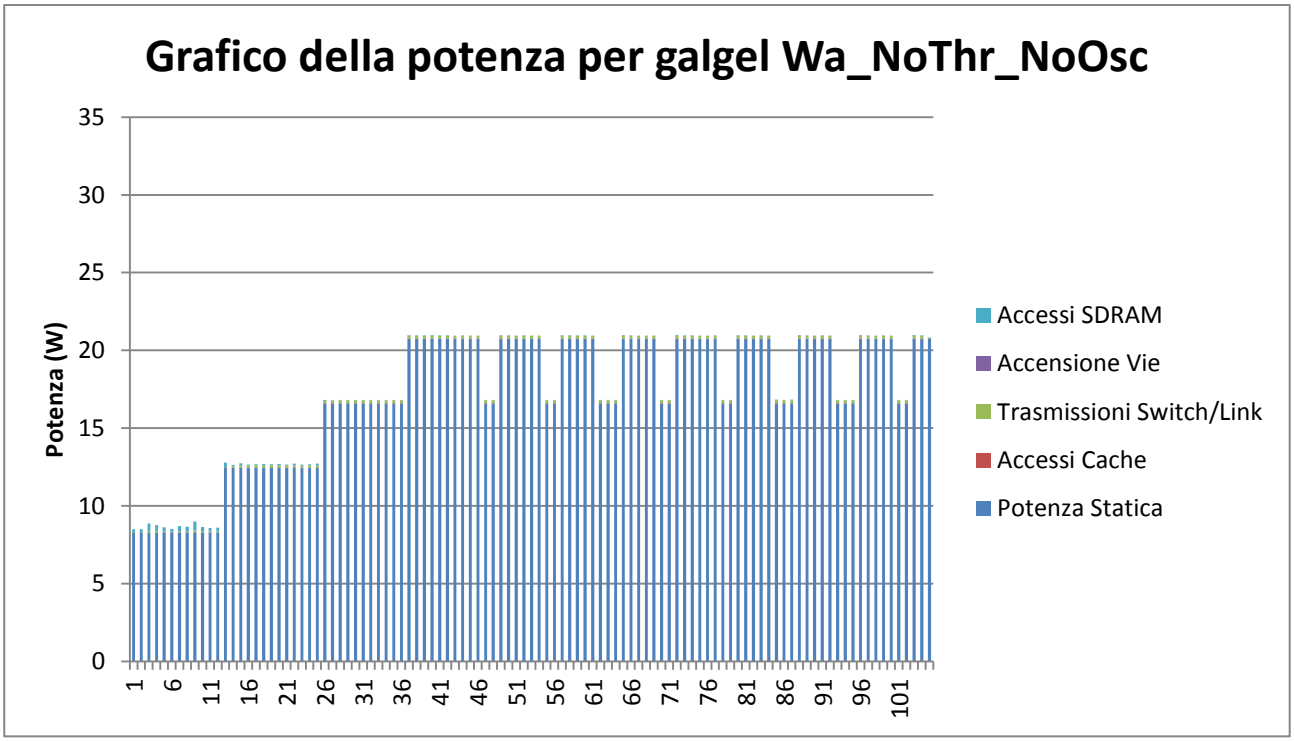


Grafico 13 : Grafico della potenza di galgel D-NUCA Wa_NoThr_NoOsc

Cache	Valore
Plain	33,62426 W
Wa_AvgThr	20,97003 W
Wa_NoThr_NoOsc	20,9844 W

Tabella 15 : Peak Power per galgel

2.1.2.8. Mesa

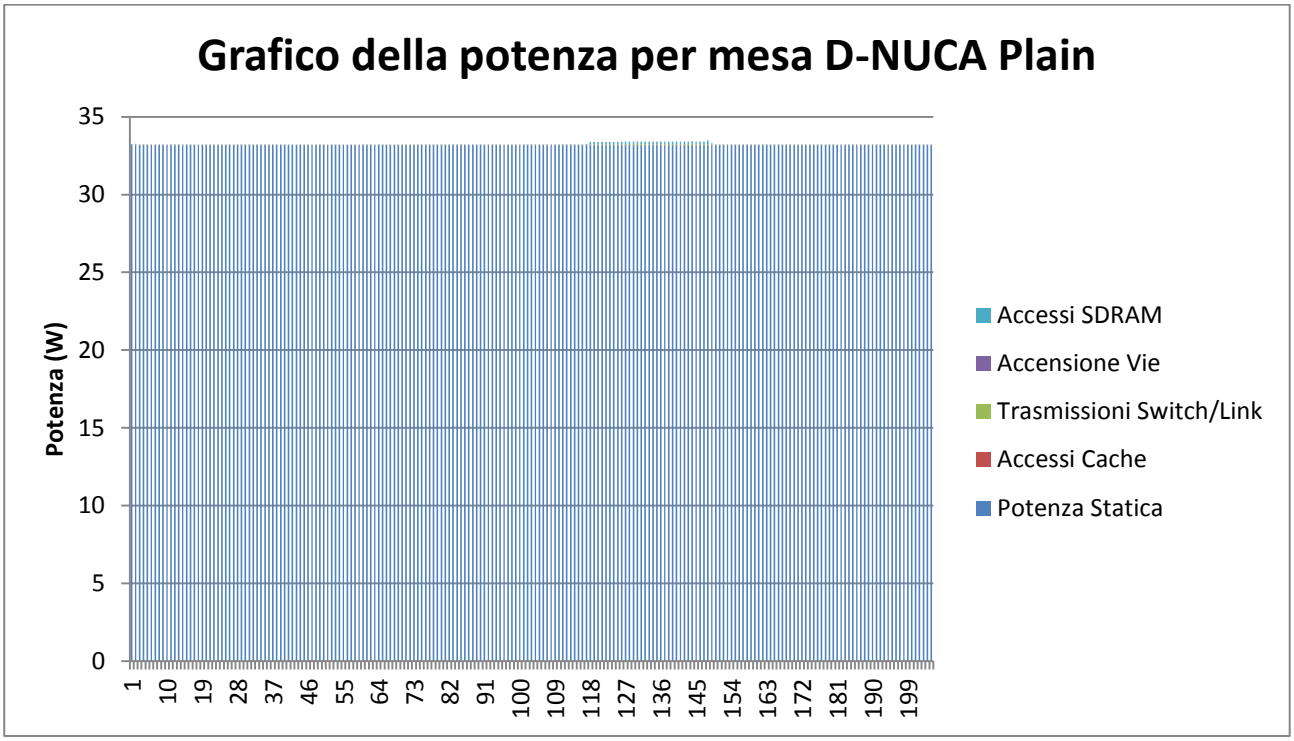


Grafico 14 : Grafico della potenza di mesa D-NUCA Plain

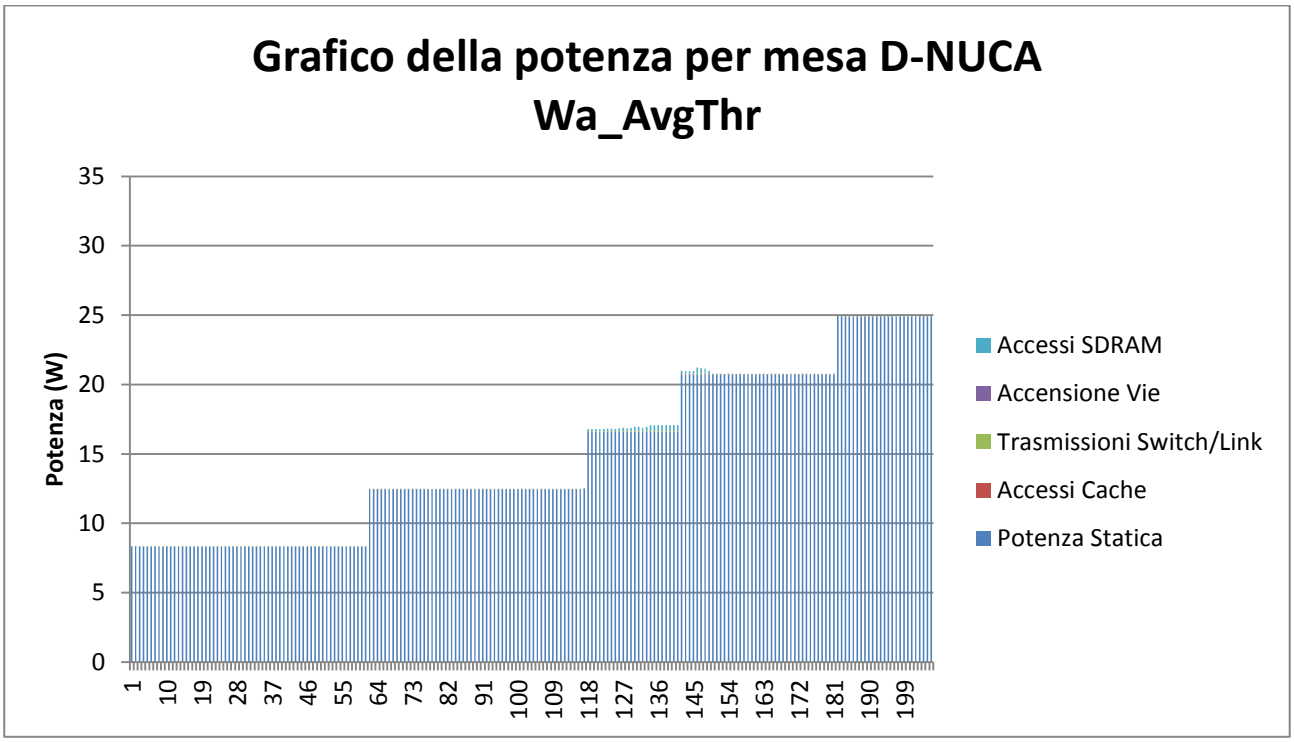


Grafico 15 : Grafico della potenza di mesa D-NUCA Wa_AvgThr

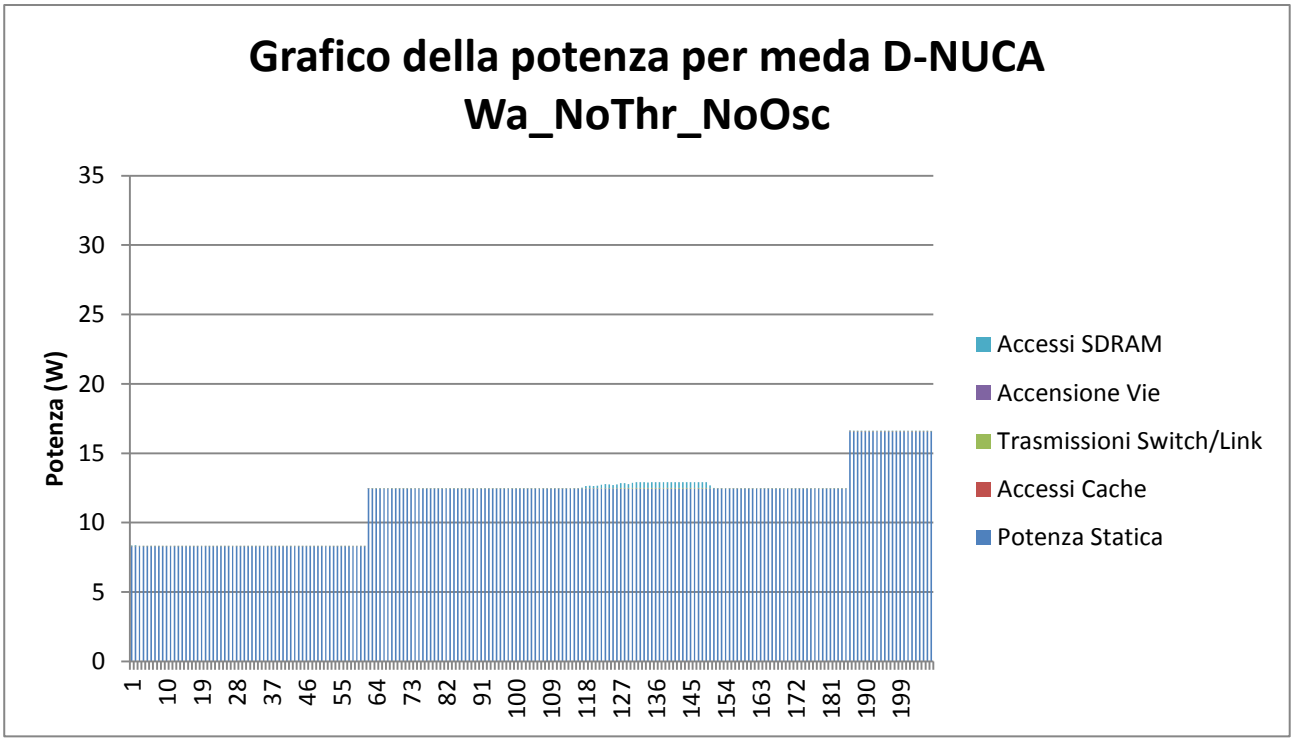


Grafico 16 : Grafico della potenza di mesa D-NUCA Wa_NoThr_NoOsc

Cache	Valore
Plain	33,48348 W
Wa_AvgThr	24,91145 W
Wa_NoThr_NoOsc	16,61271 W

Tabella 16 : Peak Power per mesa

2.1.2.9. Mgrid

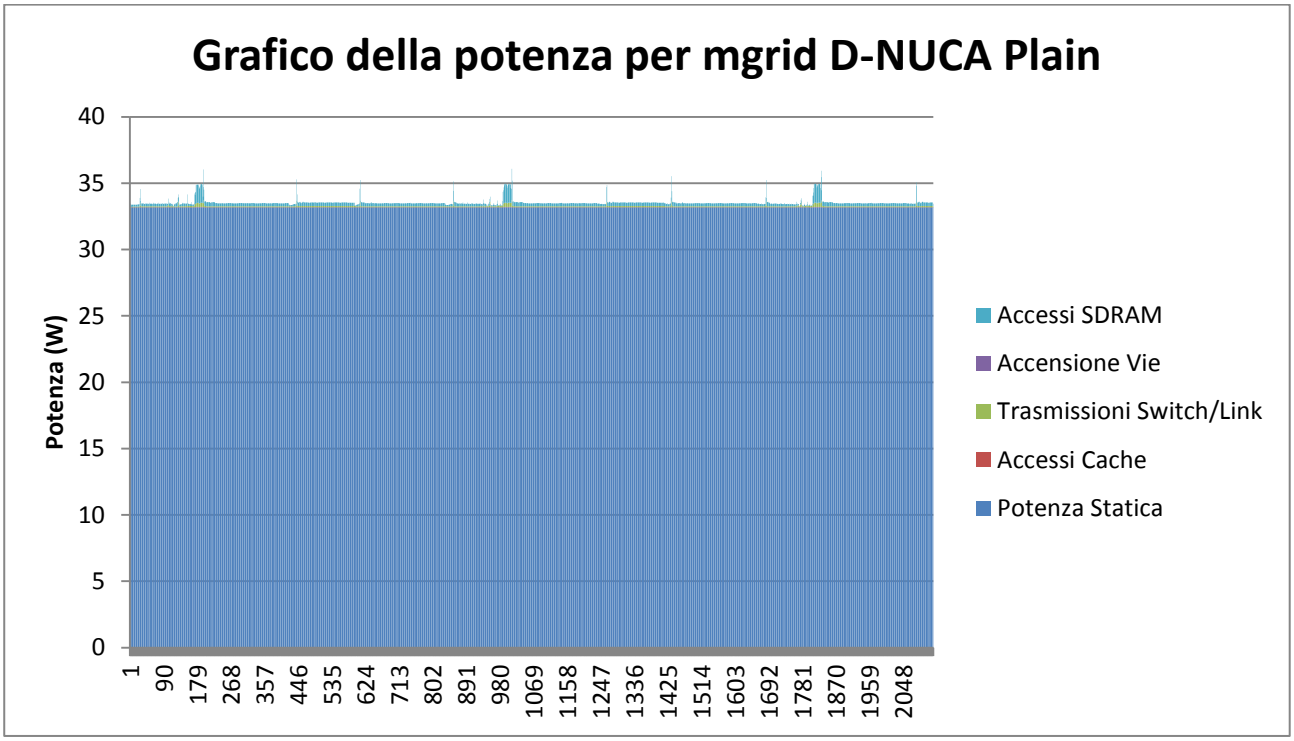


Grafico 17 : Grafico della potenza di mesa D-NUCA Wa_NoThr_NoOsc

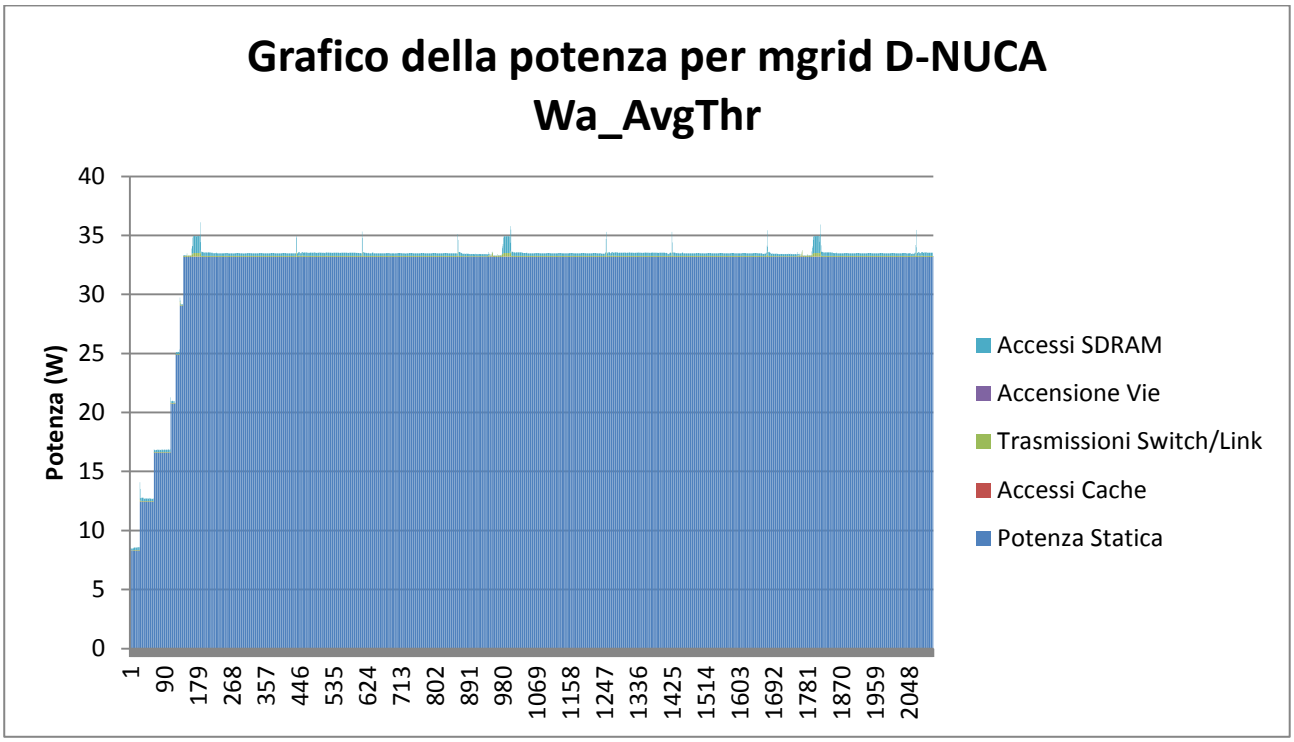


Grafico 18 : Grafico della potenza di mesa D-NUCA Wa_AvgThr

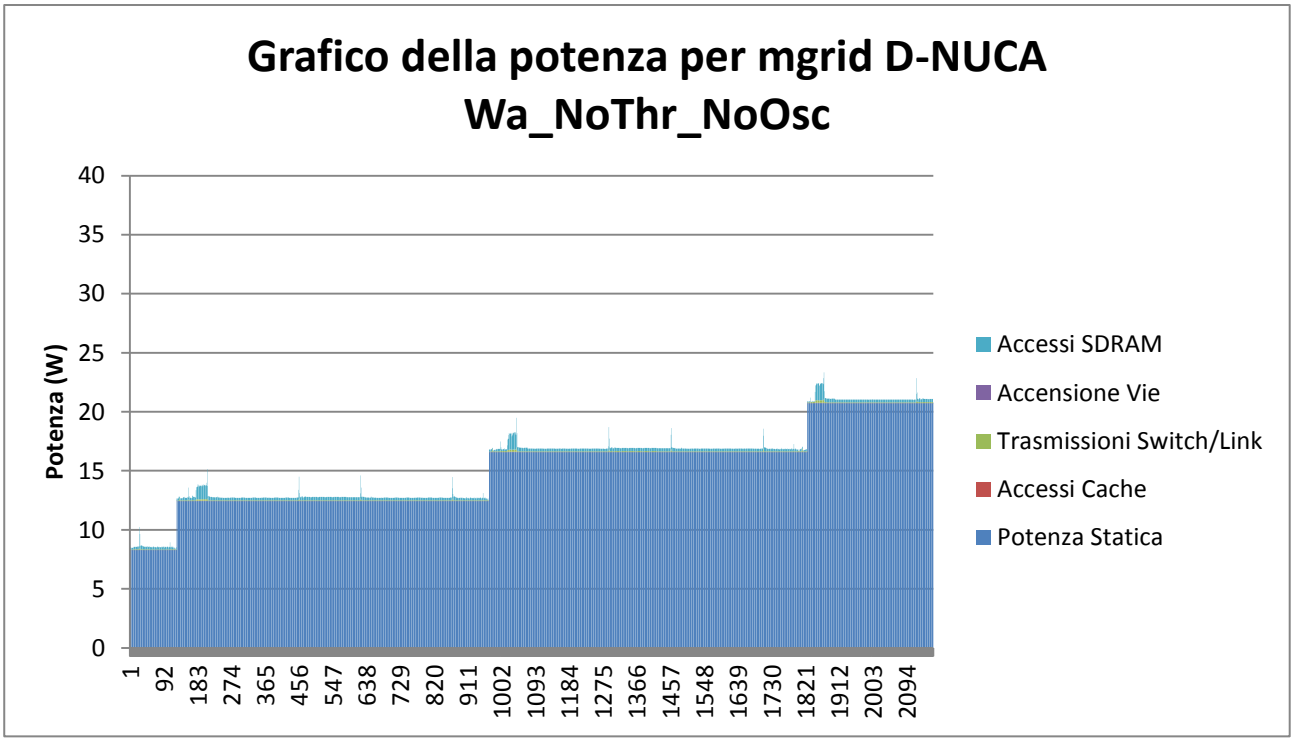


Grafico 19 : Grafico della potenza di mesa D-NUCA Wa_NoThr_NoOsc

Cache	Valore
Plain	36,09134 W
Wa_AvgThr	36,1054 W
Wa_NoThr_NoOsc	23,35154 W

Tabella 17 : Peak Power per mesa

2.1.3. Analisi Dual Core

Per effettuare l'analisi in modalità dual core , sono state simulate le diverse configurazione della cache prese in esame , utilizzando un sottoinsieme delle combinazioni dei benchmark in Tabella 1. L'insieme è stato scelto facendo riferimento al livello di associatività media delle simulazioni stesse. In particolare , sono stati creati tre intervalli basandoci sulla somma delle associatività medie calcolate per ciascun benchmark lanciate su un sistema single core. Per ciascuno di essi sono state scelte dei benchmark significativi. Il risultato è riportato in

Groppo 1 (WA1 + WA2 < 8)		**Gruppo 2 (WA1+WA2 ~= 8)**		**Gruppo 3 (WA1 + WA2 > 8)**	
Core 1	**Core 2**	**Core 1**	**Core 2**	**Core 1**	**Core 2**
Twolf	Twolf	Mesa	Mesa	Mgrid	Perlbmk
Perlbmk	Perlbmk	Art	Galgel	Equake	Galgel
Art	Perlbmk	Galgel	Mesa	Mgrid	Galgel

Andiamo ad analizzare i risulatati prodotti da ciascuna simulazione,separate per gruppi

2.1.3.1. Gruppo 1

2.1.3.1.1. twolf_twolf

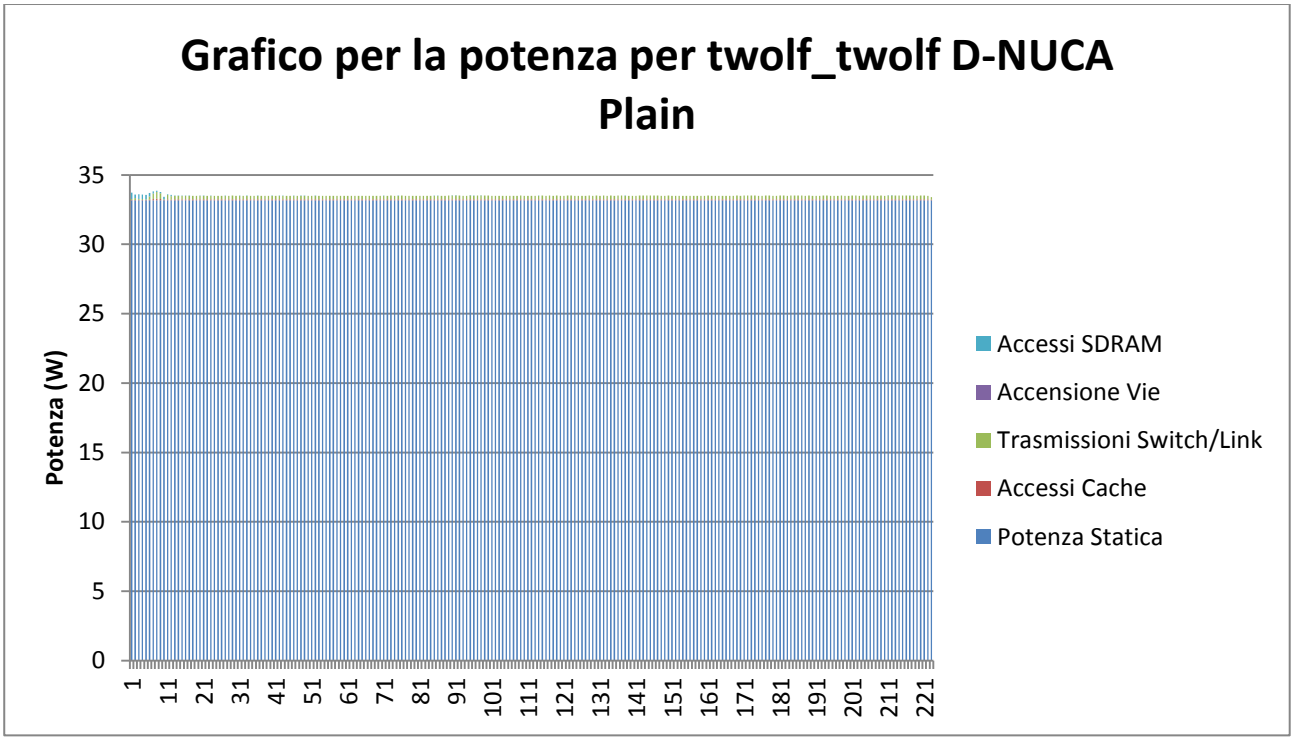


Grafico 20 : Grafico della potenza di twolf_twolf D-NUCA Plain

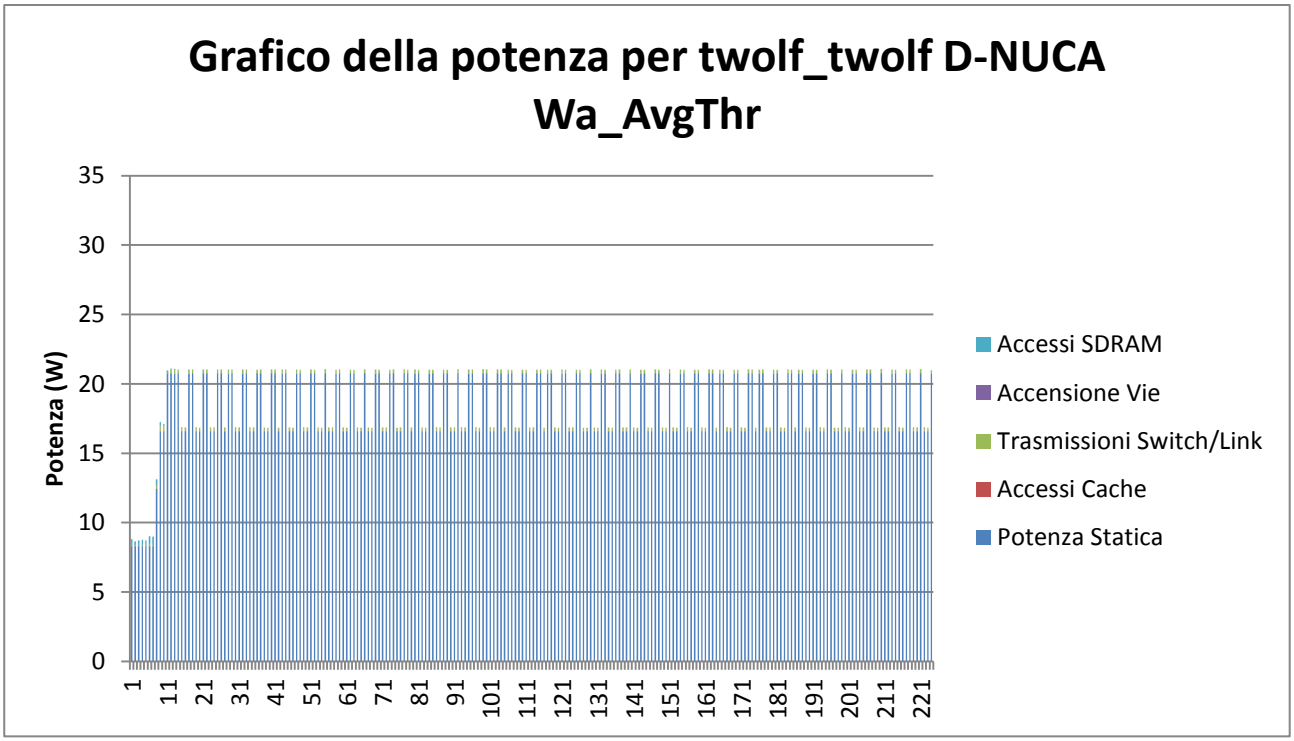


Grafico 21 : Grafico della potenza di twolf_twolf D-NUCA Wa_AvgThr

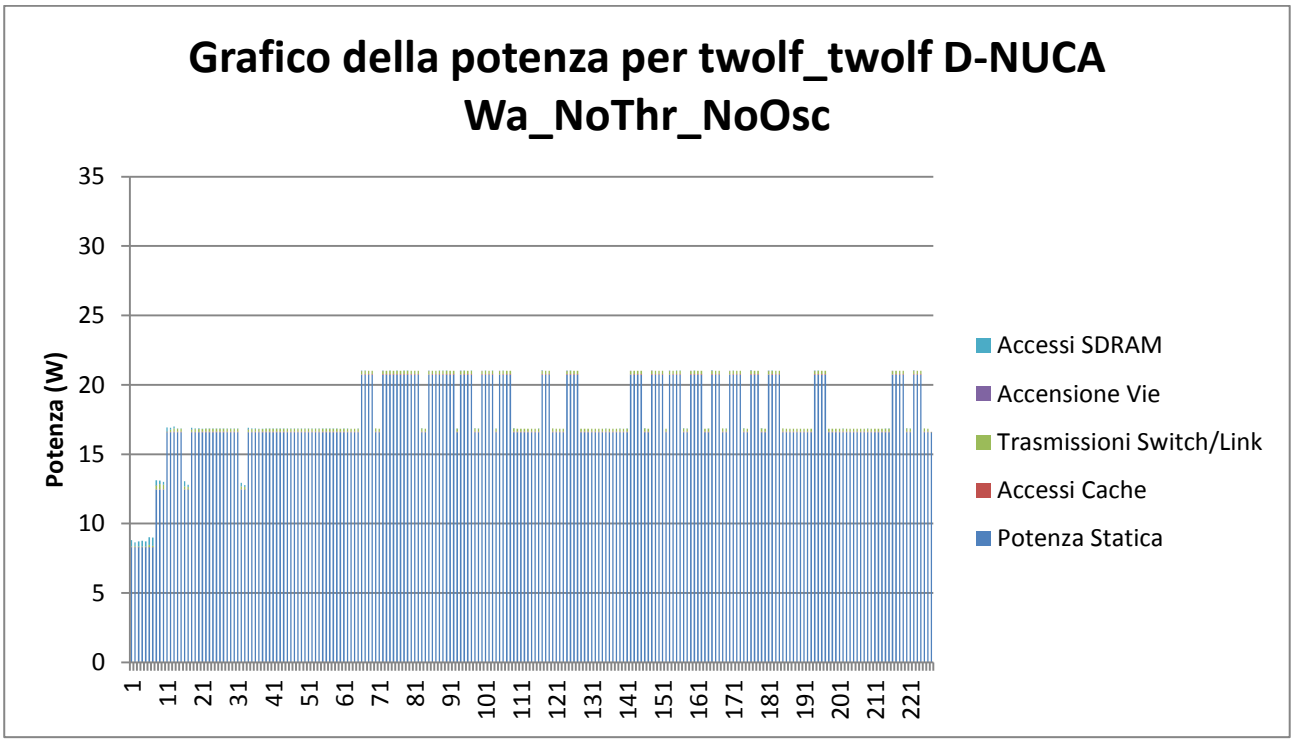


Grafico 22 : Grafico della potenza di twolf_twolf D-NUCA Wa_NoThr_NoOsc

Cache	Valore
Plain	33,8591 W
Wa_AvgThr	21,10478 W
Wa_NoThr_NoOsc	21,01303 W

Tabella 18 : Peak Power per twolf_twolf

2.1.3.1.2. *perlbmk_perlbmk*

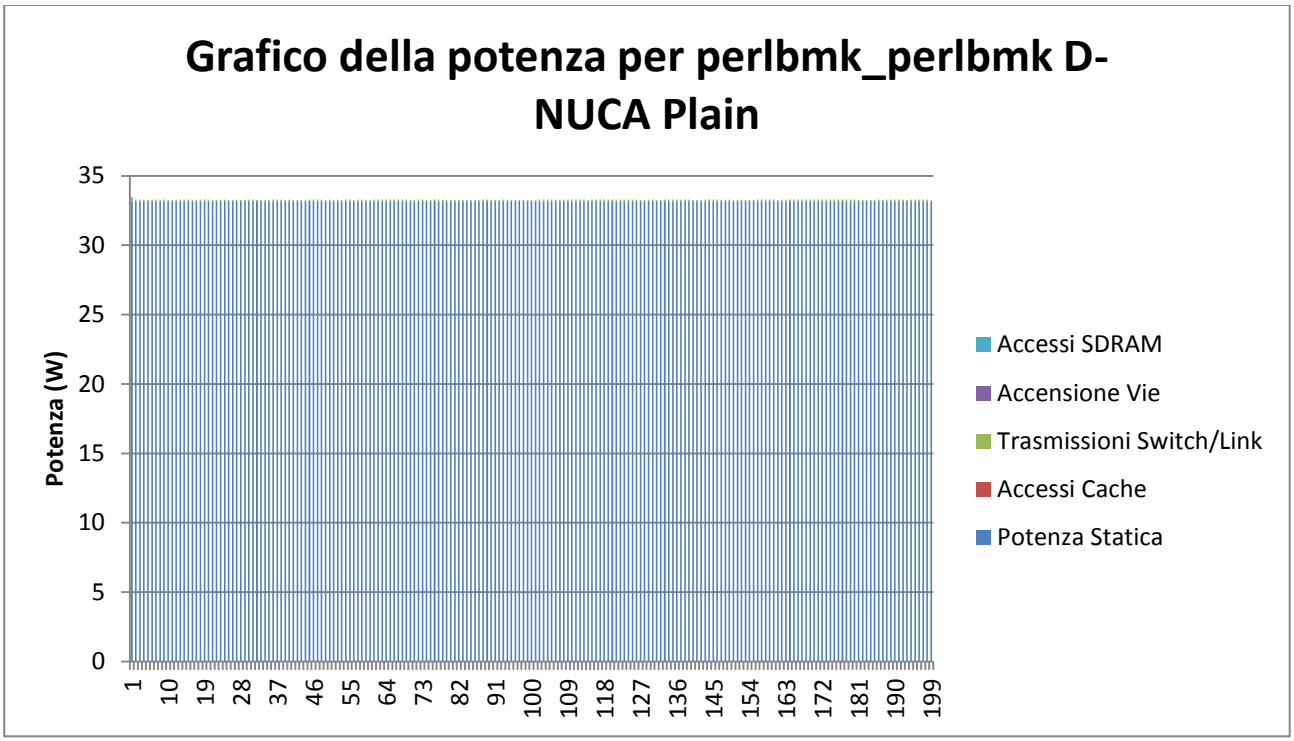


Grafico 23 : Grafico della potenza di perlbmk_perlbmk D-NUCA Plain

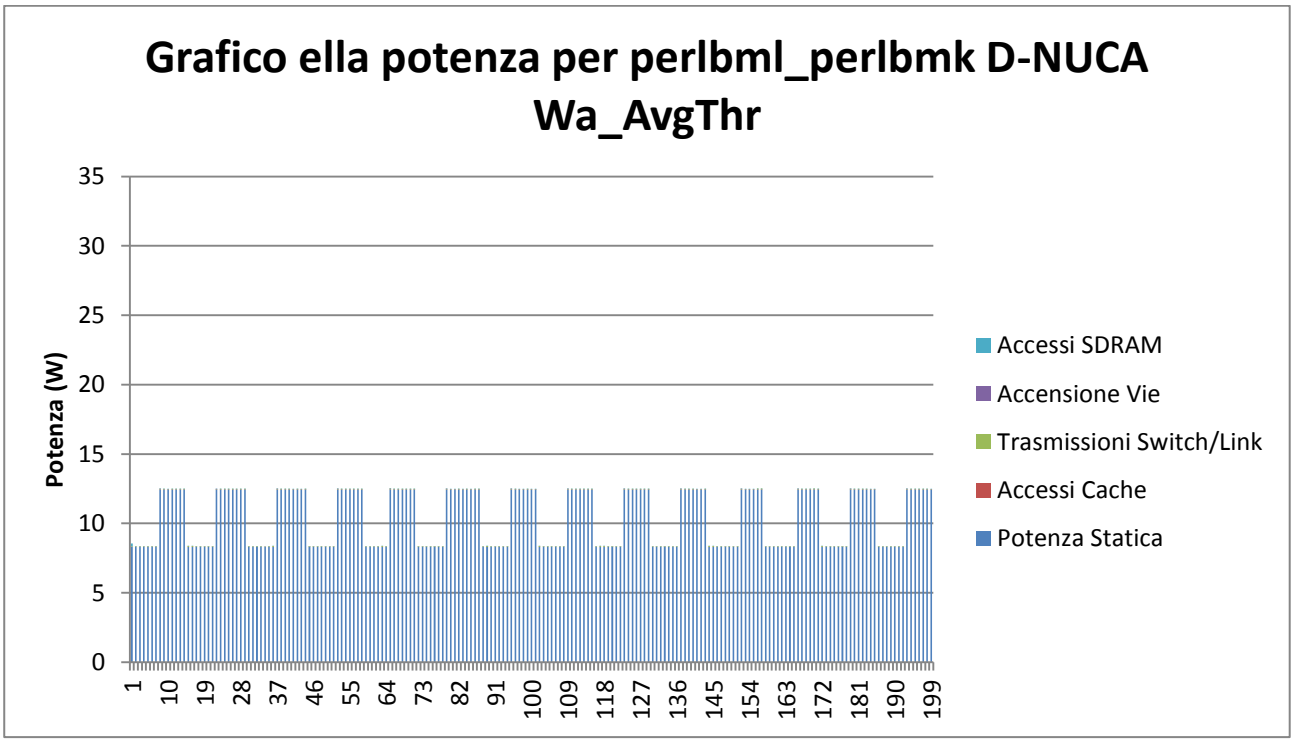


Grafico 24 : Grafico della potenza di perlbmk_perlbmk D-NUCA wa_AvgThr

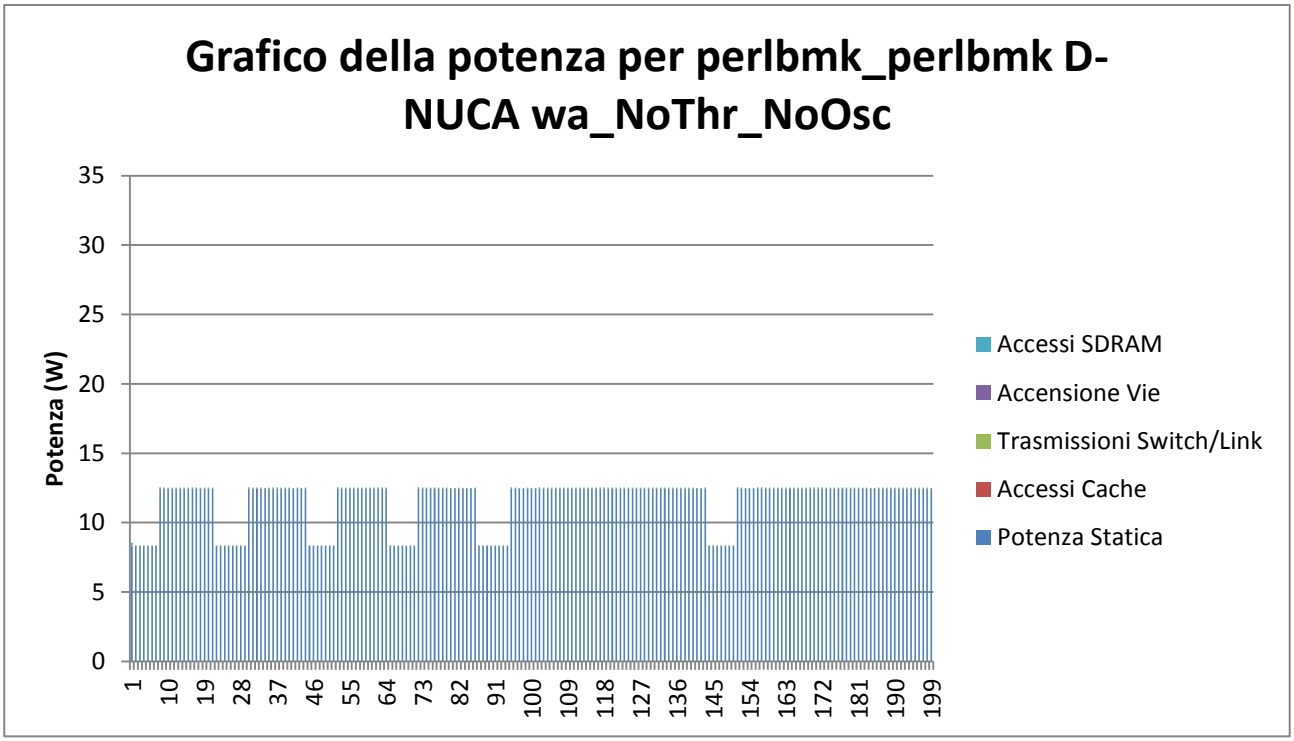


Grafico 25 : Grafico della potenza di perlbmk_perlbmk D-NUCA Wa_NoThr_NoOsc

Cache	Valore
Plain	33,45569 W
Wa_AvgThr	12,51999 W
Wa_NoThr_NoOsc	12,52096 W

Tabella 19 : Peak Power per perlbmk_perlbmk

2.1.3.1.3. *art_perlbmk*

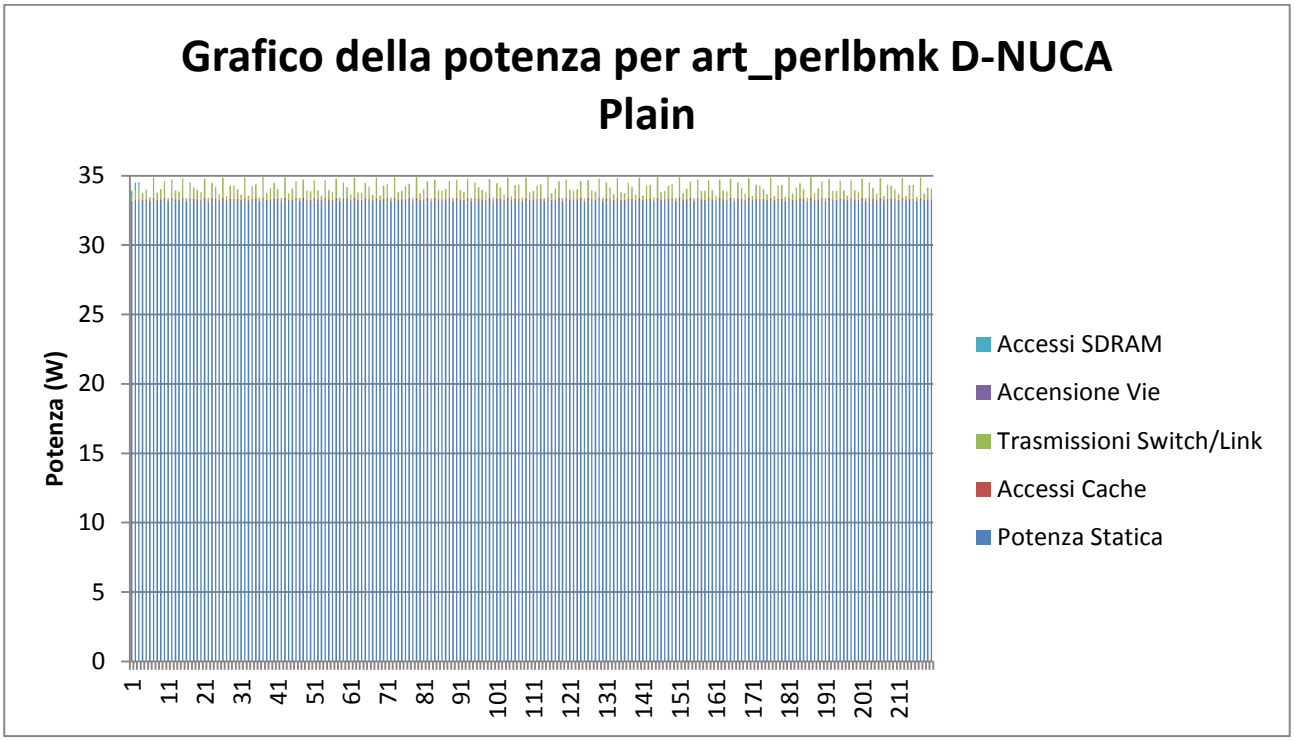


Grafico 26 : Grafico della potenza di art_perlbmk D-NUCA Plain

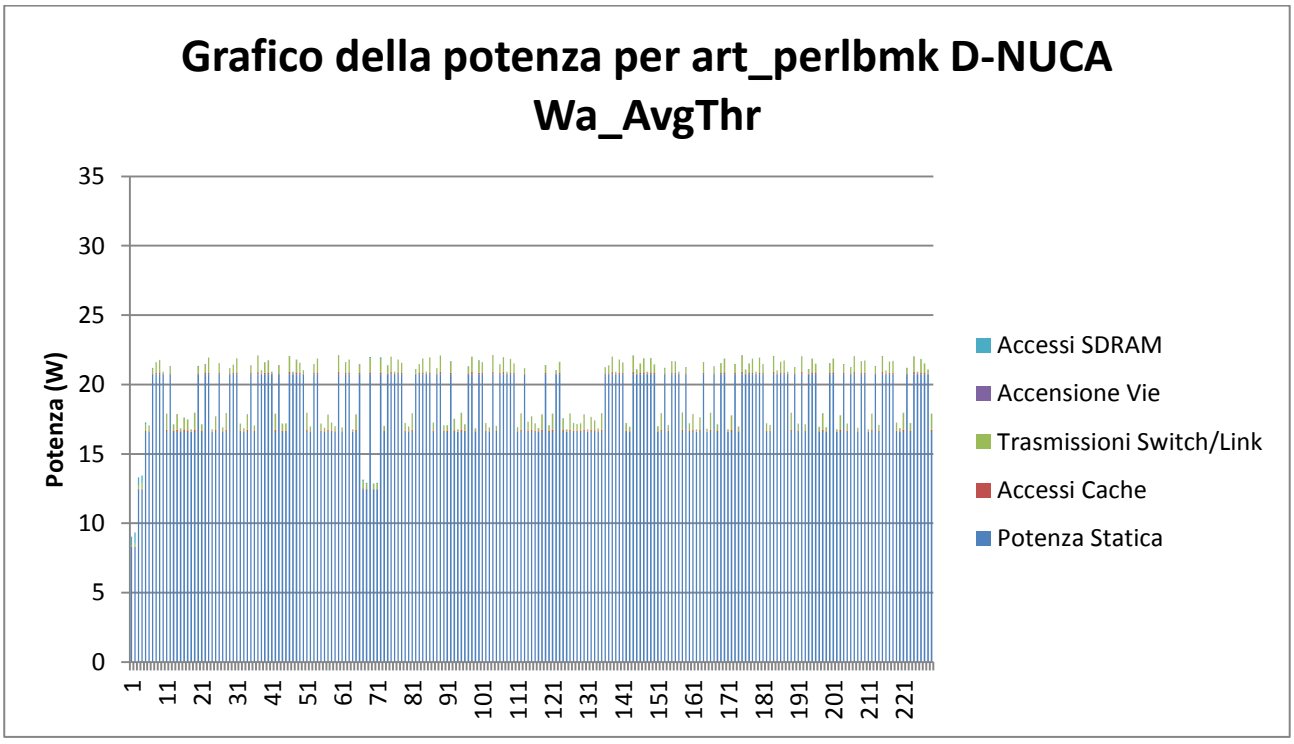


Grafico 27 : Grafico della potenza di art_perlbmk D-NUCA Wa_AvgThr

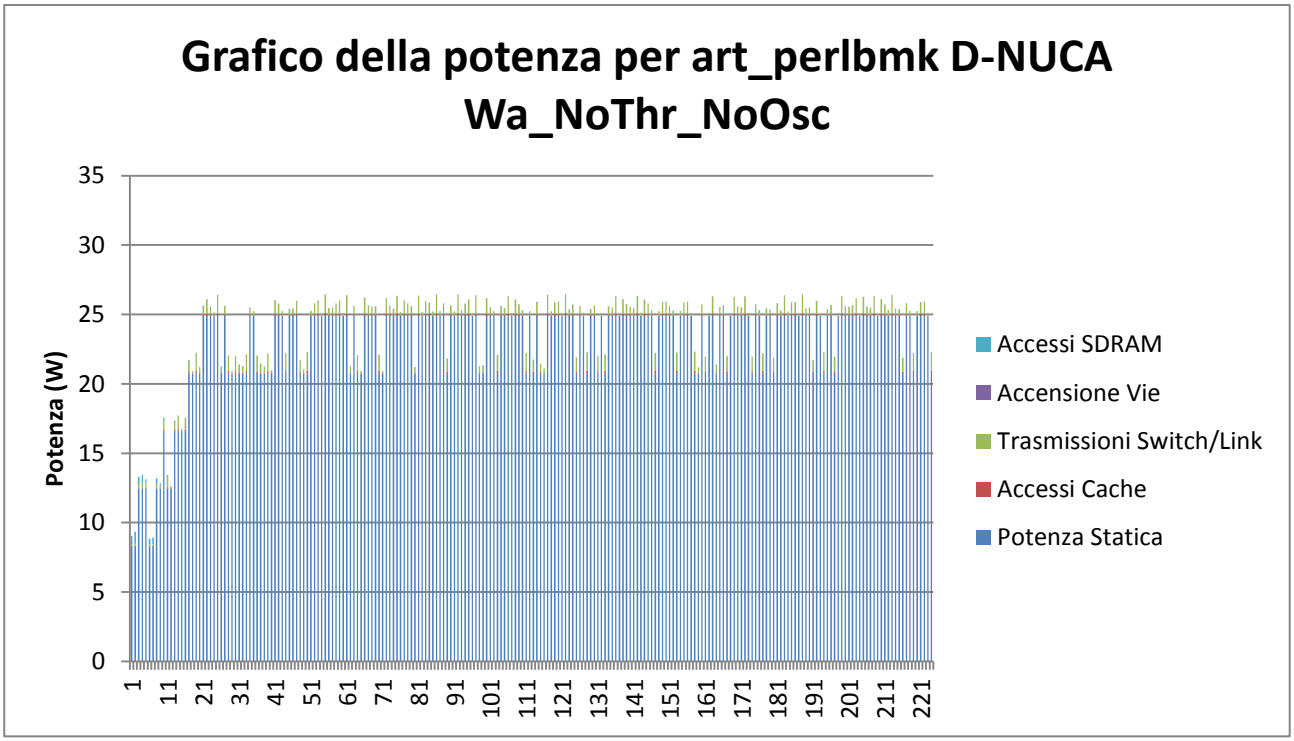


Grafico 28 : Grafico della potenza di art_perlbmk D-NUCA Wa_NoThr_NoOsc

Cache	Valore
Plain	34,89634 W
Wa_AvgThr	22,10416 W
Wa_NoThr_NoOsc	26,44598 W

Tabella 20 : Peak Power per art_perlbmk

2.1.3.2. Gruppo 2

2.1.3.1.4. Mesa_mesa

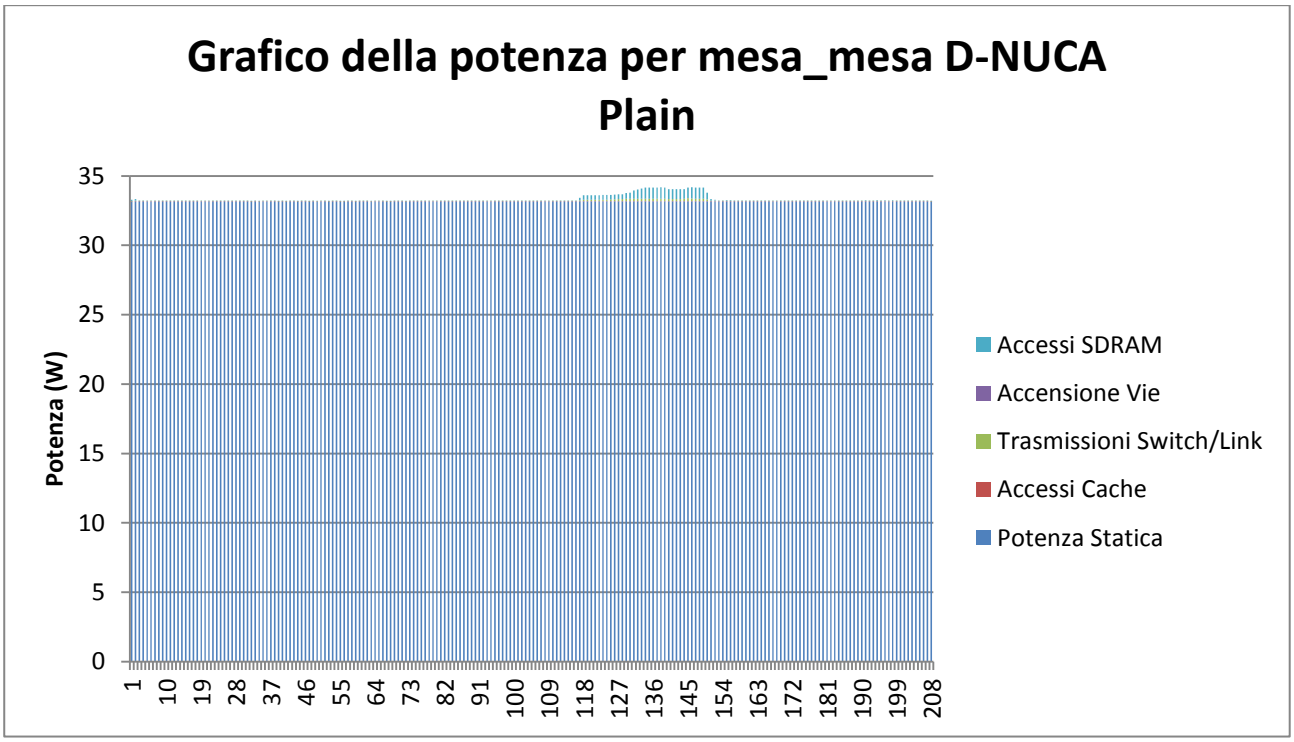


Grafico 29 : Grafico della potenza di mesa_mesa D-NUCA Plain

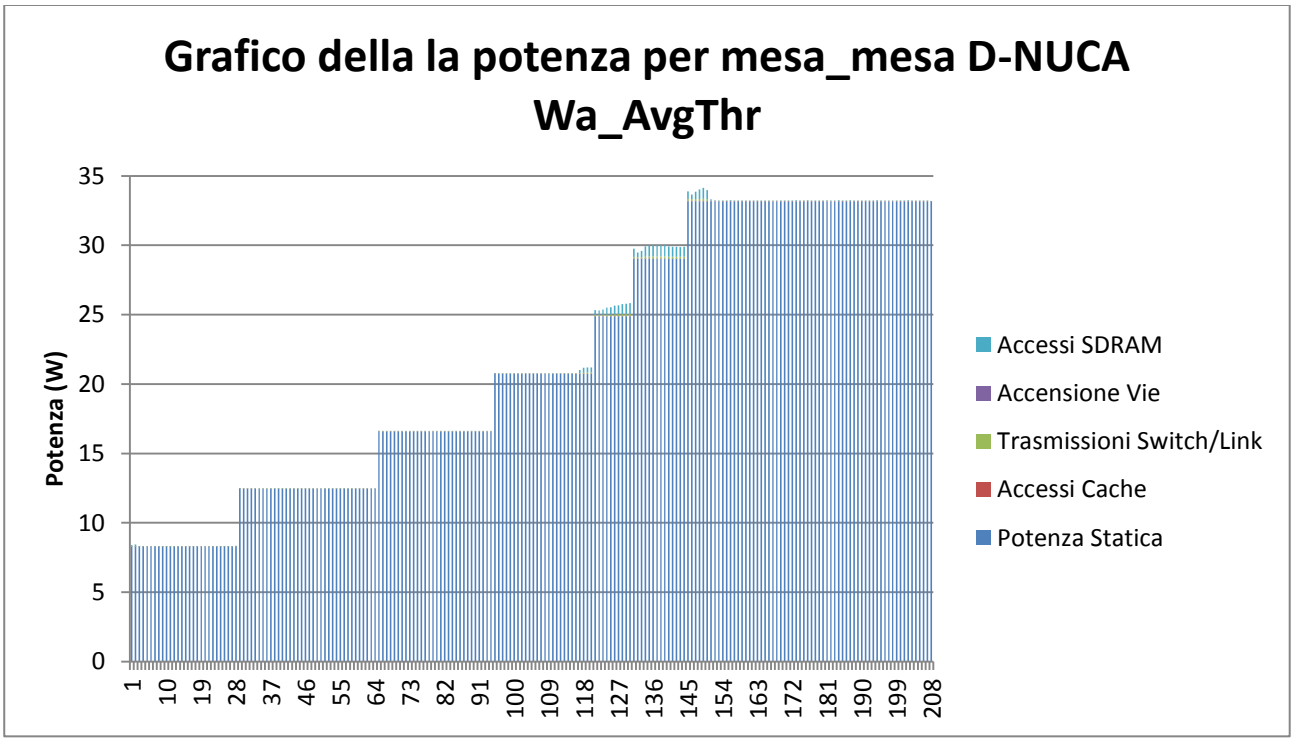


Grafico 30 : Grafico della potenza di mesa_mesa D-NUCA Wa_AvgThr

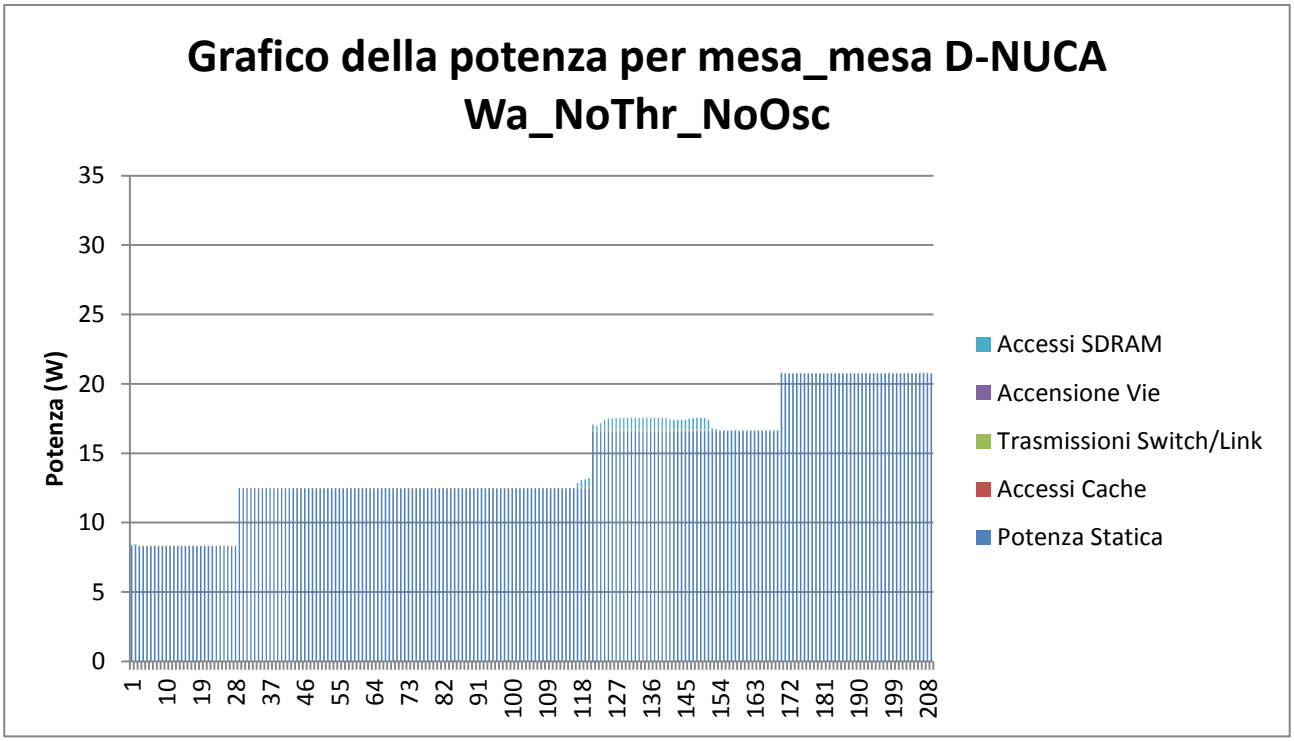


Grafico 31 : Grafico della potenza di mesa_mesa D-NUCA Wa_NoThr_NoOsc

Cache	Valore
Plain	34,17943 W
Wa_AvgThr	34,12976W
Wa_NoThr_NoOsc	20,79835 W

Tabella 21 : Peak Power per mesa_mesa

2.1.3.1.5. *Art_gelgel*

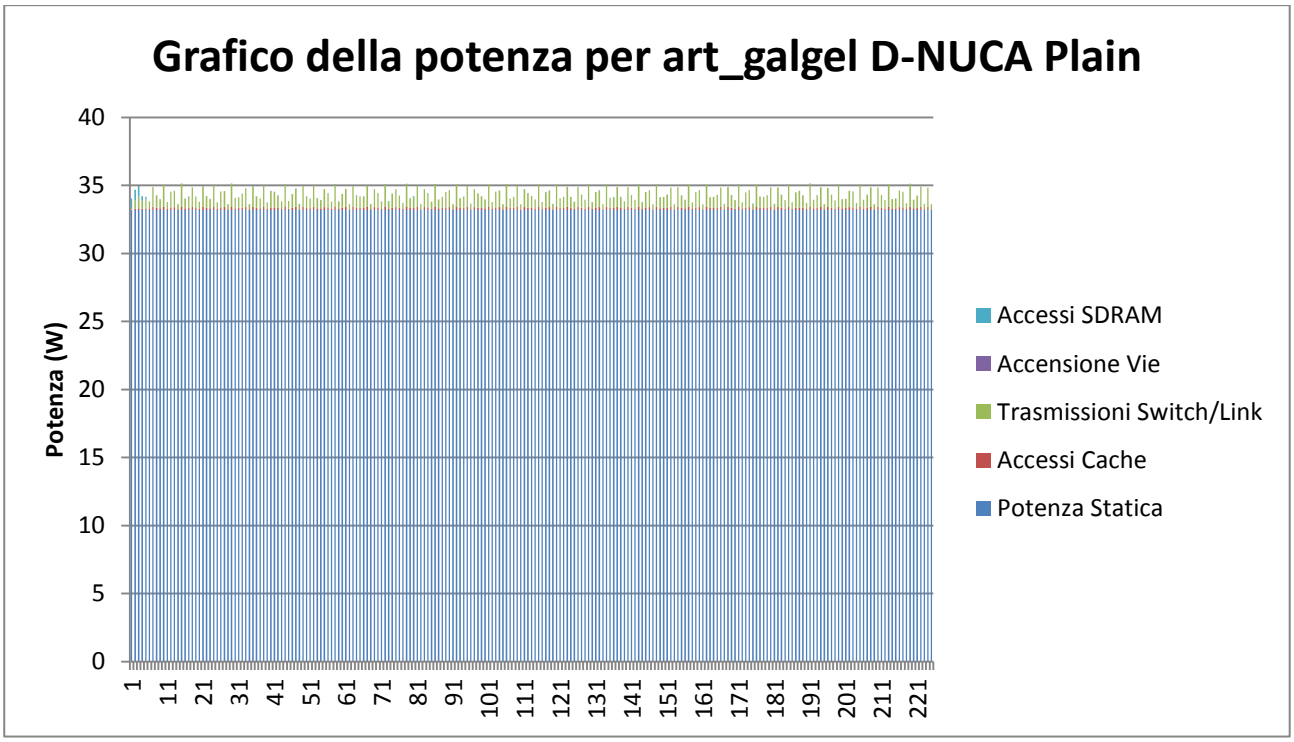


Grafico 32 : Grafico della potenza di art_galgel D-NUCA Plain

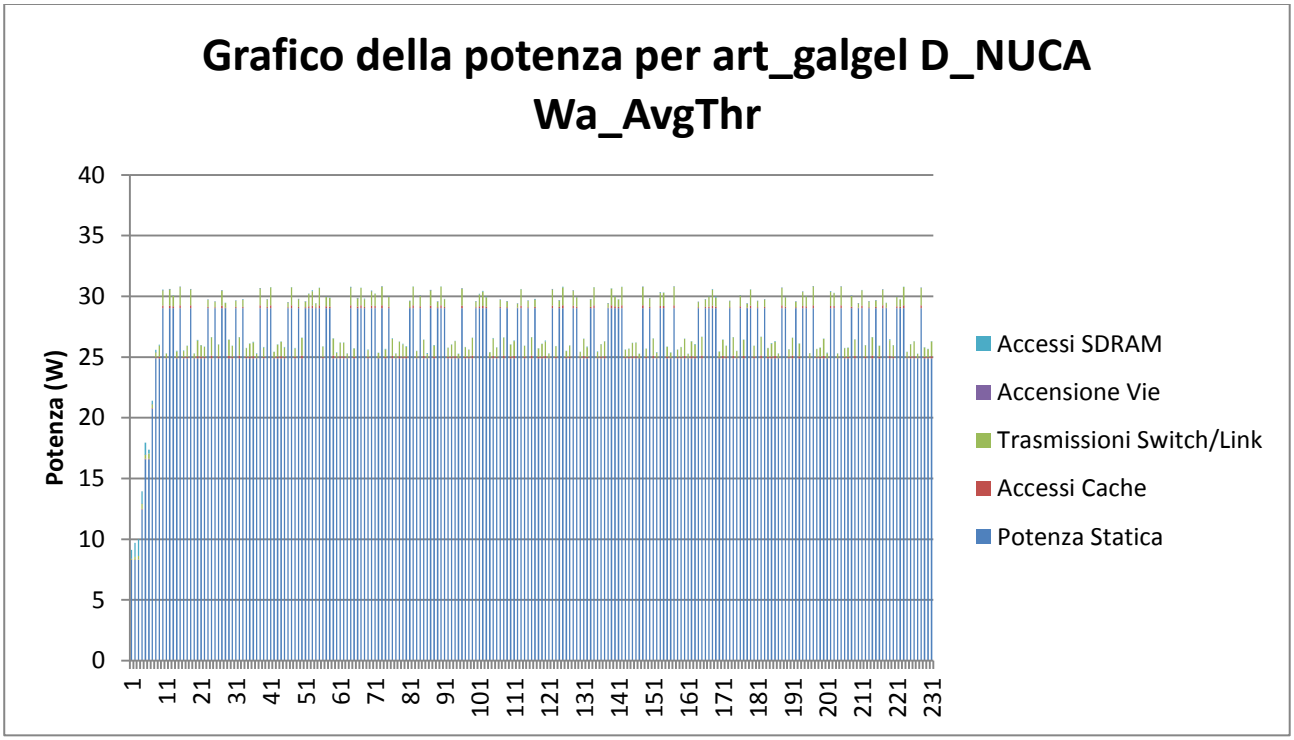


Grafico 33 : Grafico della potenza di art_galgel D-NUCA Wa_AvgThr

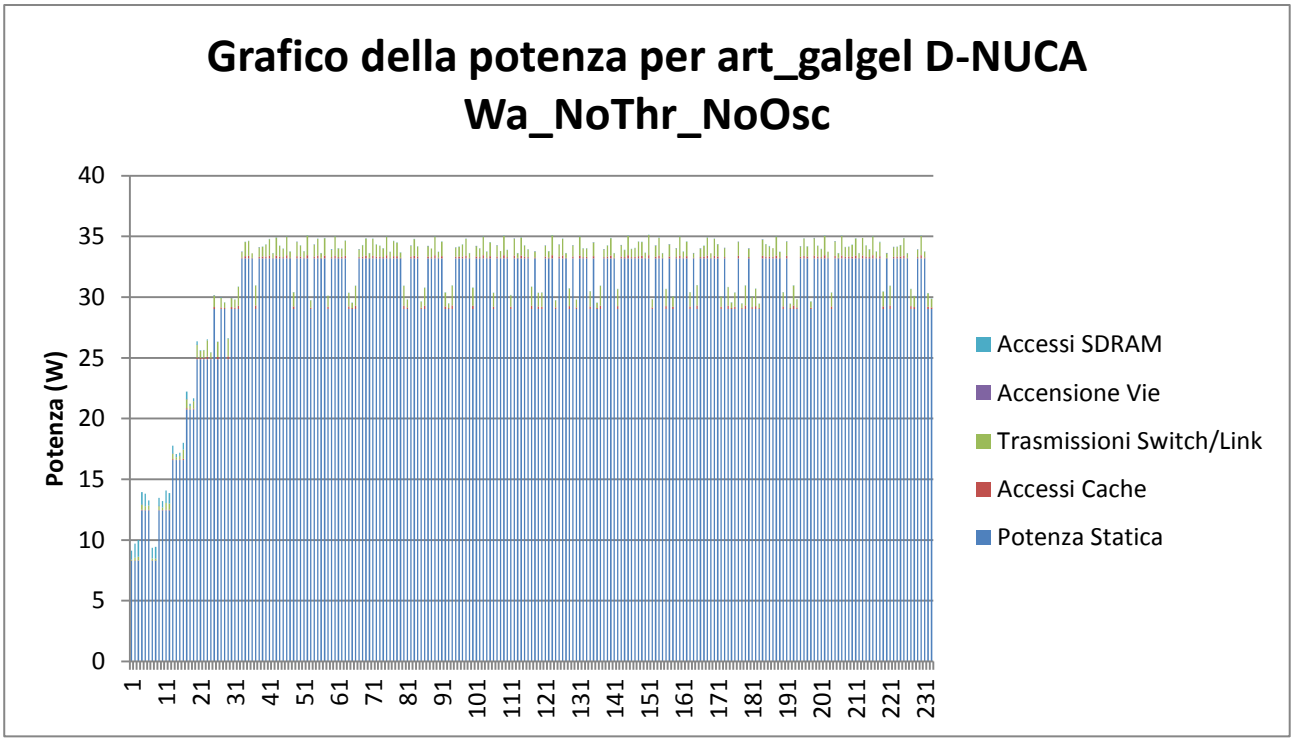


Grafico 34 : Grafico della potenza di art_galgel D-NUCA Wa_NoThr_NoOsc

Cache	Valore
Plain	35,14449 W
Wa_AvgThr	30,82893 W
Wa_NoThr_NoOsc	20,79835 W

Tabella 22 : Peak Power per art_galgel

2.1.3.1.6. *galgel_mesa*

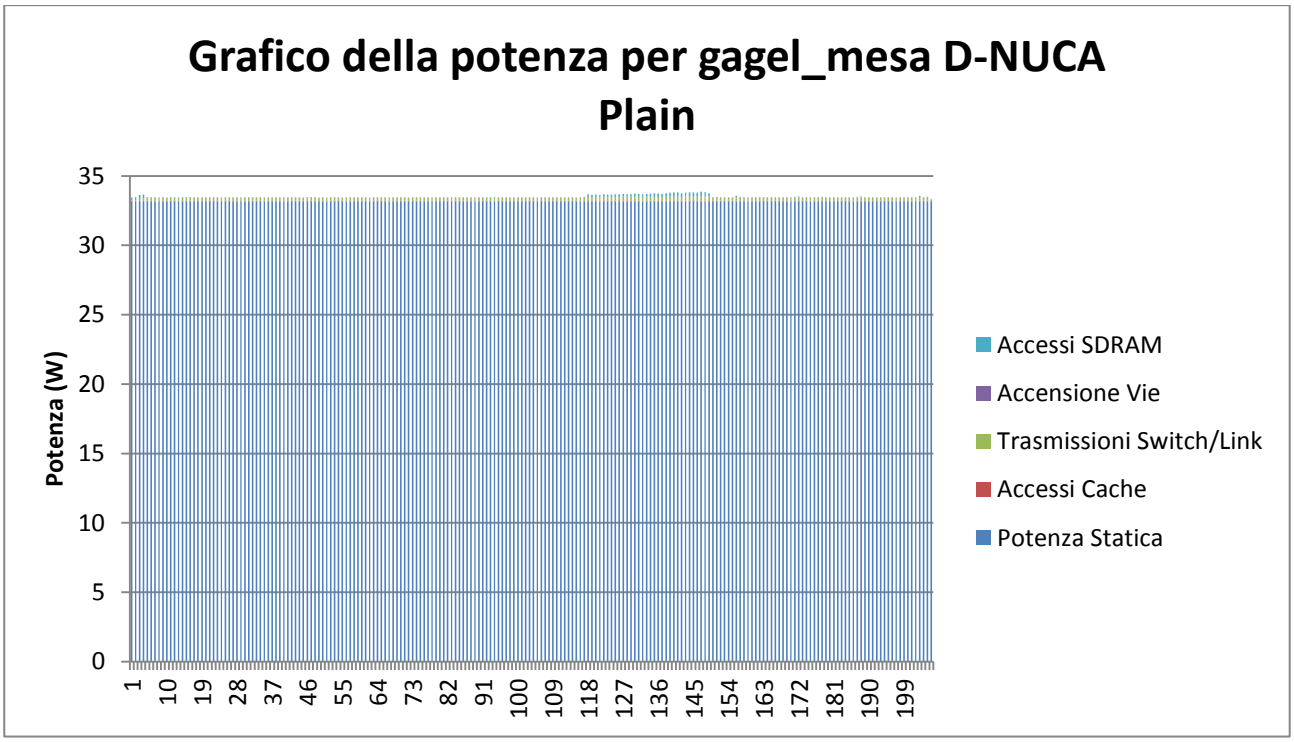


Grafico 35 : Grafico della potenza di galgel_mesa D-NUCA Plain

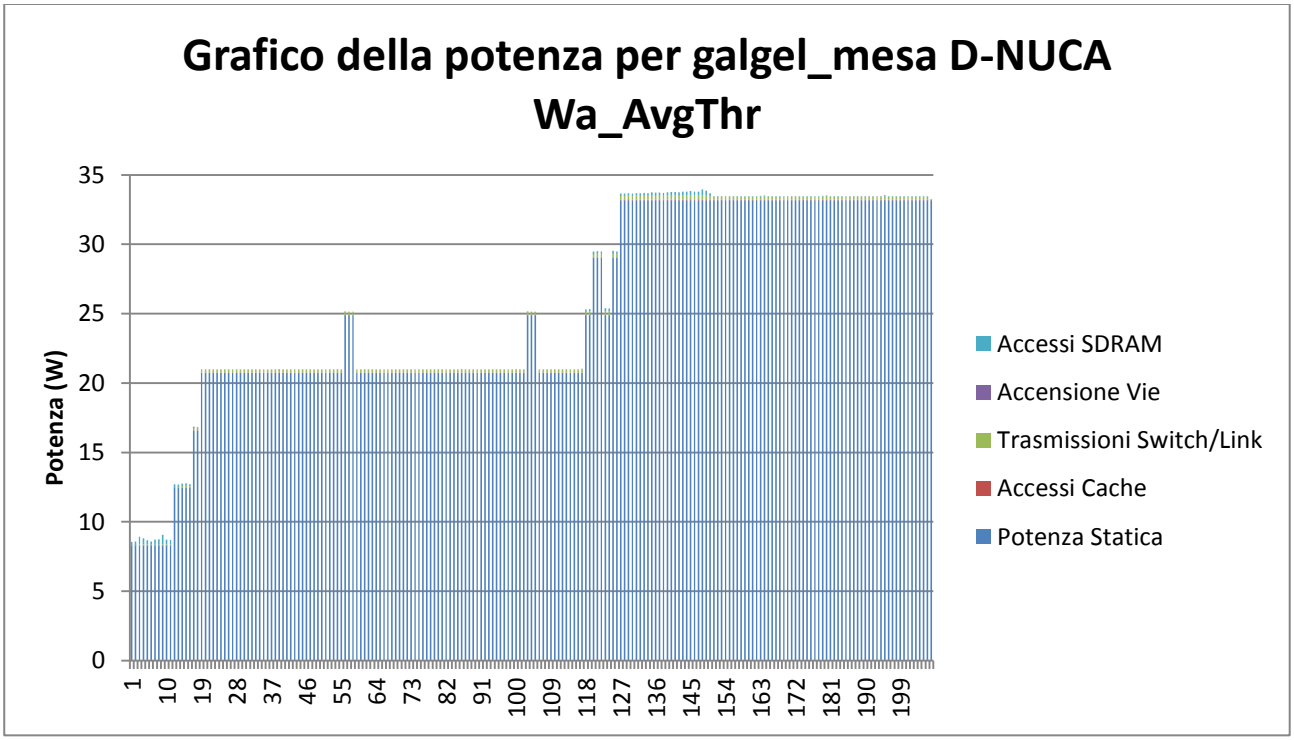


Grafico 36 : Grafico della potenza di galgel_mesa D-NUCA Wa_AvgThr

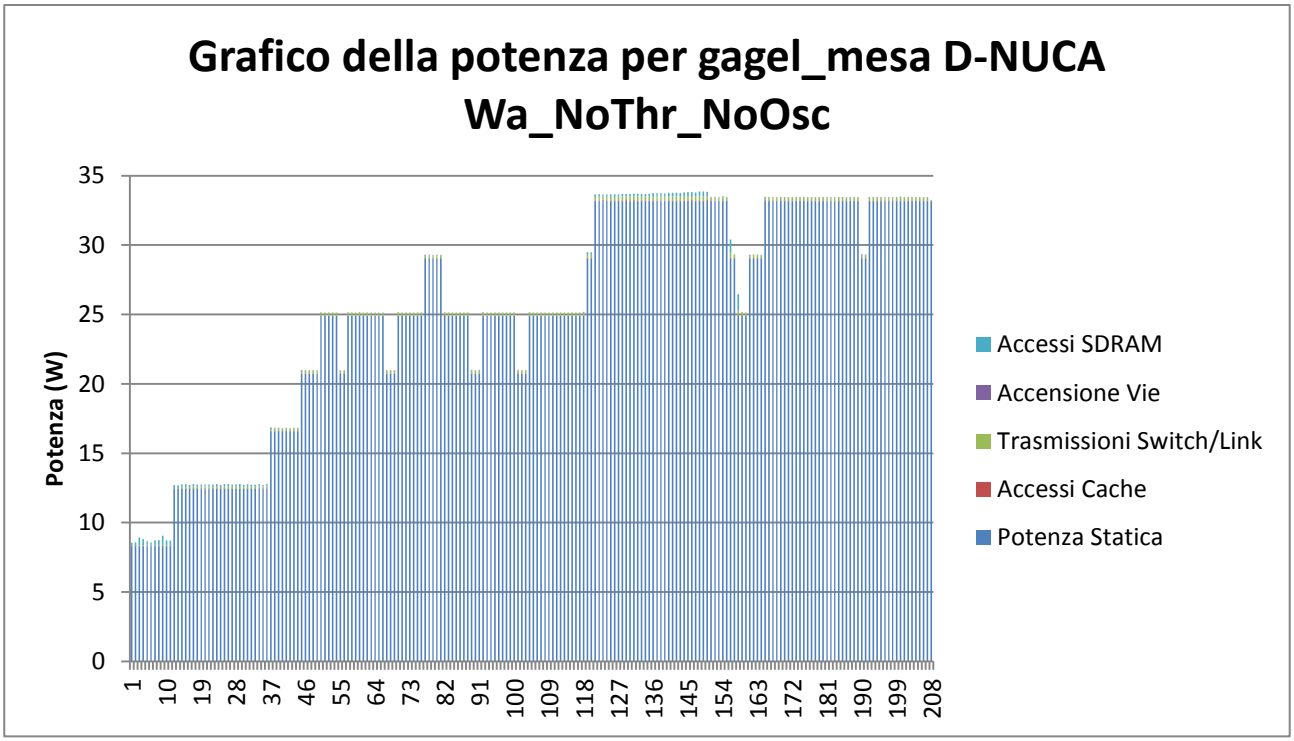


Grafico 37 : Grafico della potenza di galgel_mesa D-NUCA Wa_NoThr_NoOsc

Cache	Valore
Plain	33,86075 W
Wa_AvgThr	33,95245 W
Wa_NoThr_NoOsc	33,87208 W

Tabella 23 : Peak Power per galgel_mesa

2.1.3.3. Gruppo 3

2.1.3.1.7.

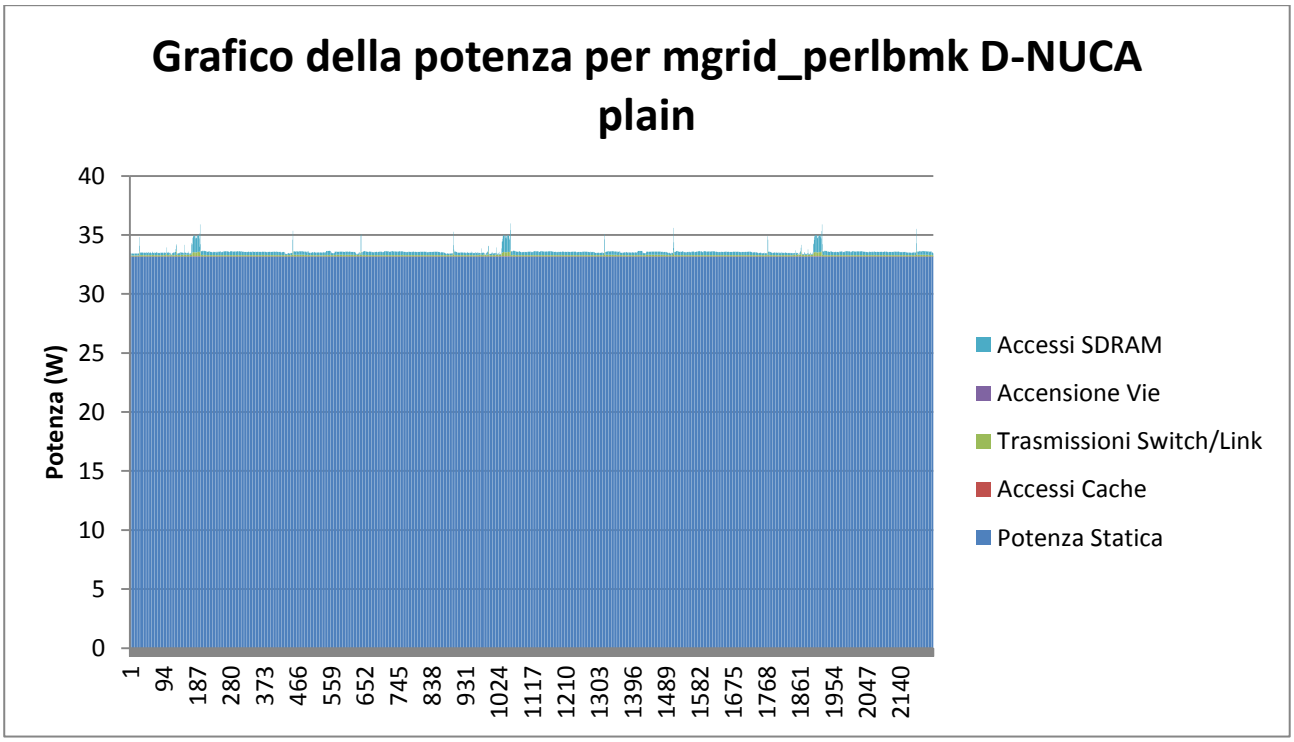


Grafico 38 : Grafico della potenza di mgrid_perlbmk D-NUCA Plain

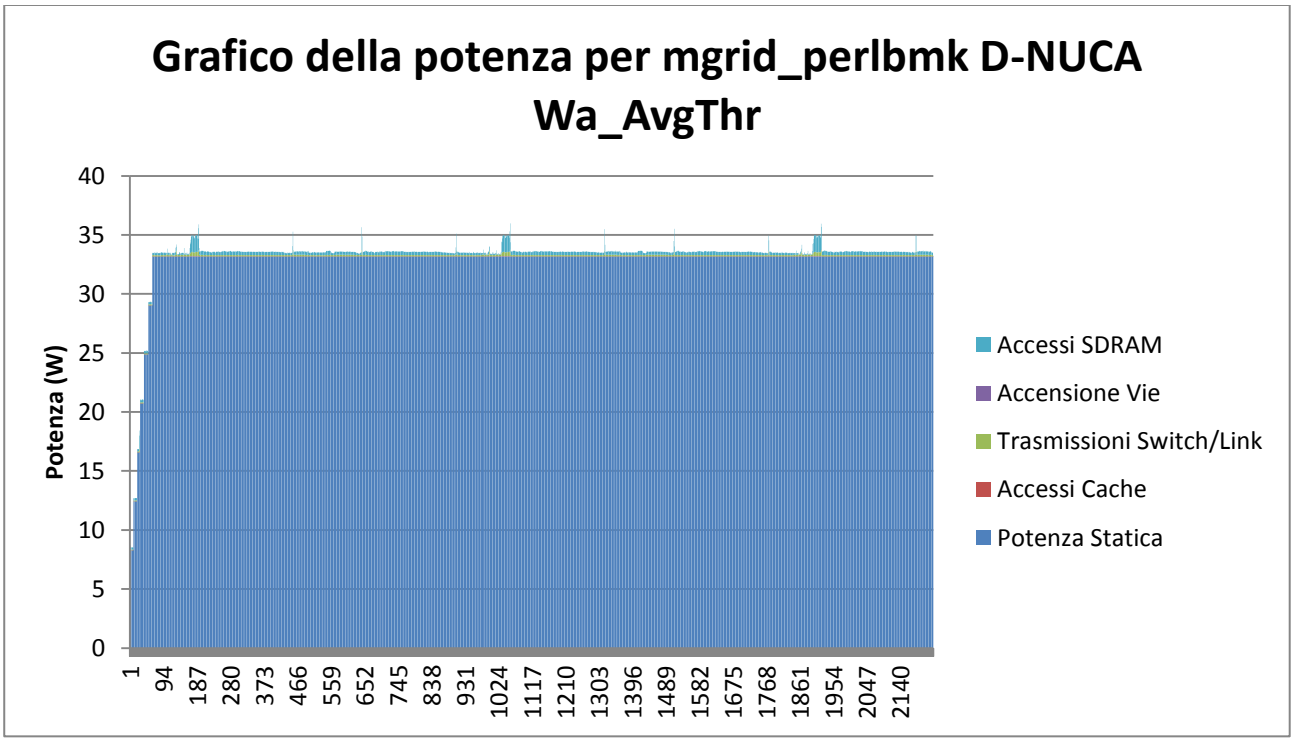


Grafico 39 : Grafico della potenza di mgrid_perlbmk D-NUCA Wa_AvgThr

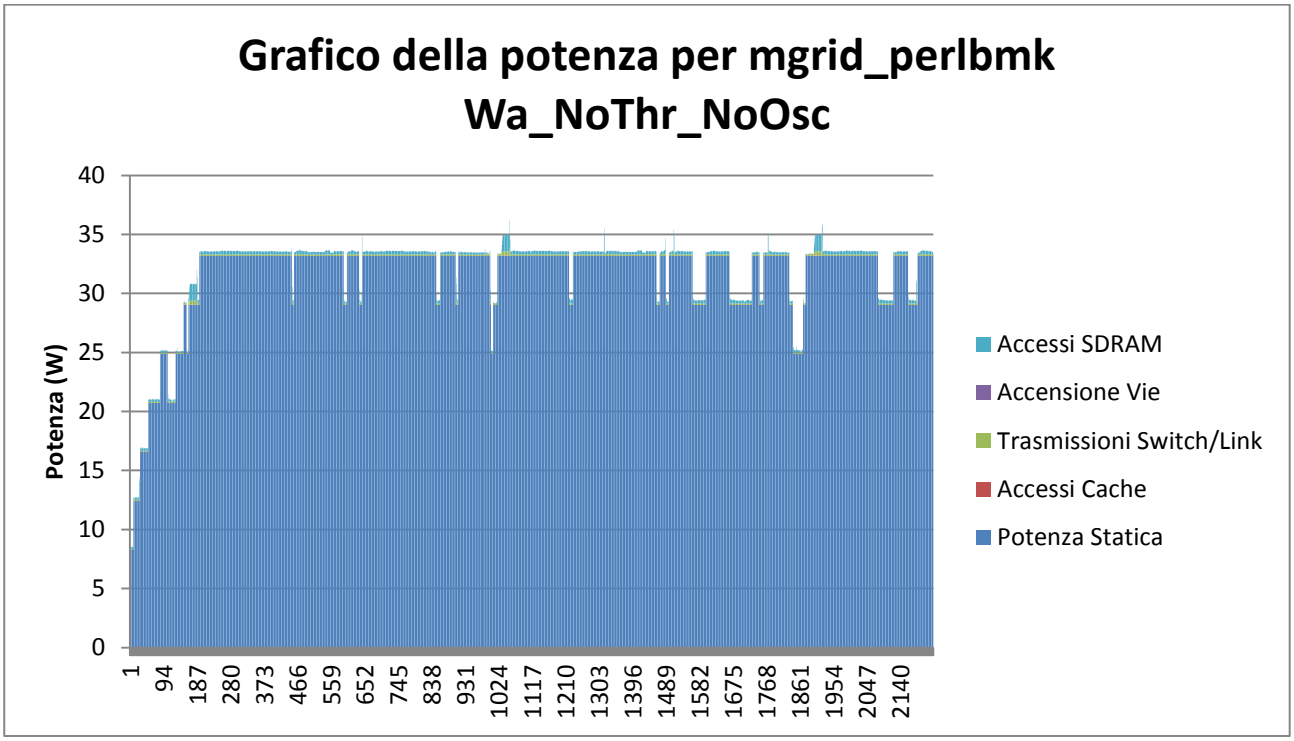


Grafico 40 : Grafico della potenza di mgrid_perlbmk D-NUCA Wa_NoThr_NoOsc

Cache	Valore
Plain	35,96804 W
Wa_AvgThr	35,98606 W
Wa_NoThr_NoOsc	36,18711 W

Tabella 24 : Peak Power per mgrid_perlbmk

2.1.3.1.8. *equake_galgel*

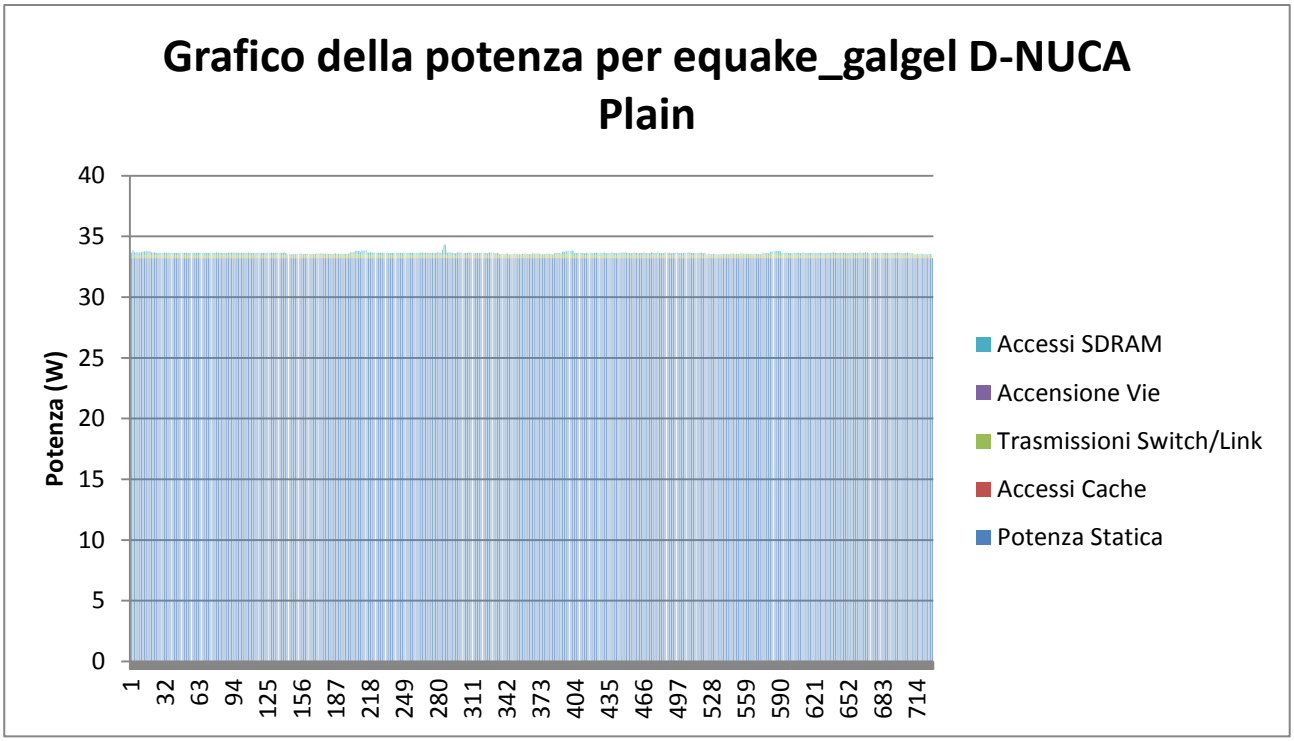


Grafico 41 : Grafico della potenza di equake_galgel D-NUCA Wa_NoThr_NoOsc

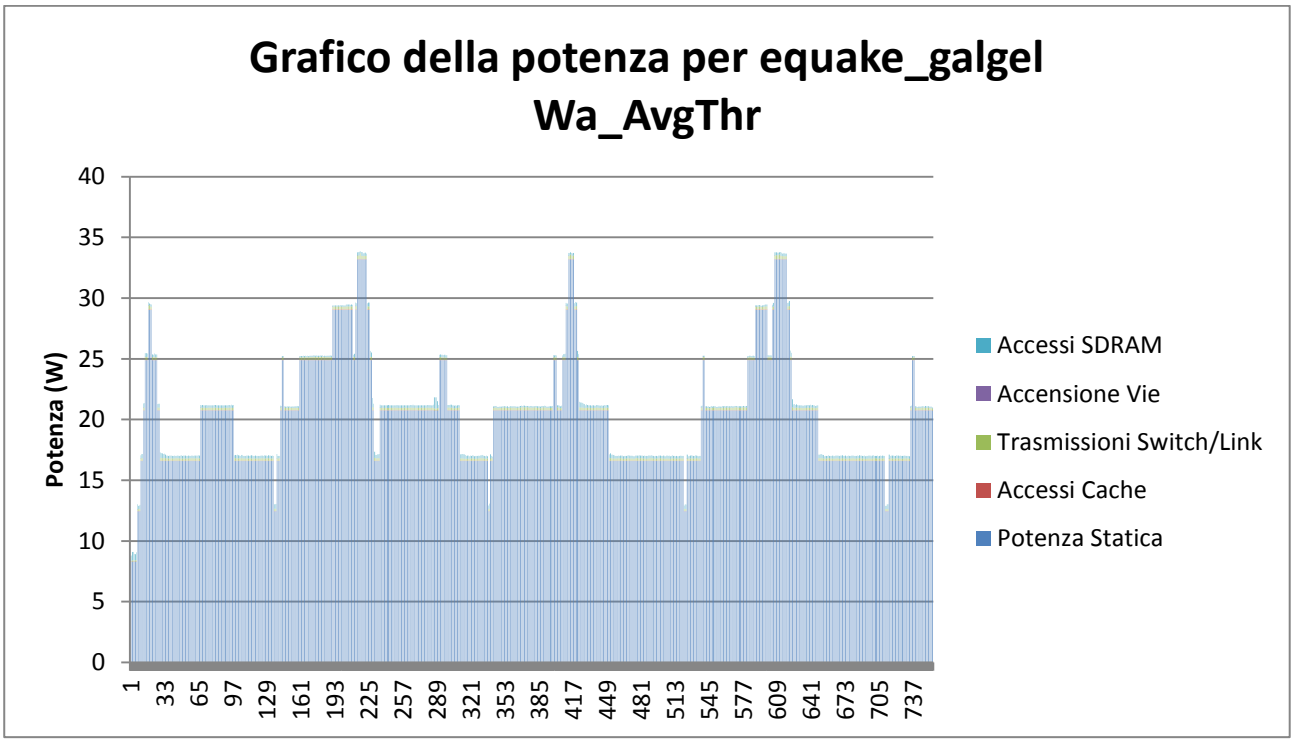


Grafico 42 : Grafico della potenza di equake_galgel D-NUCA Wa_NoThr_NoOsc

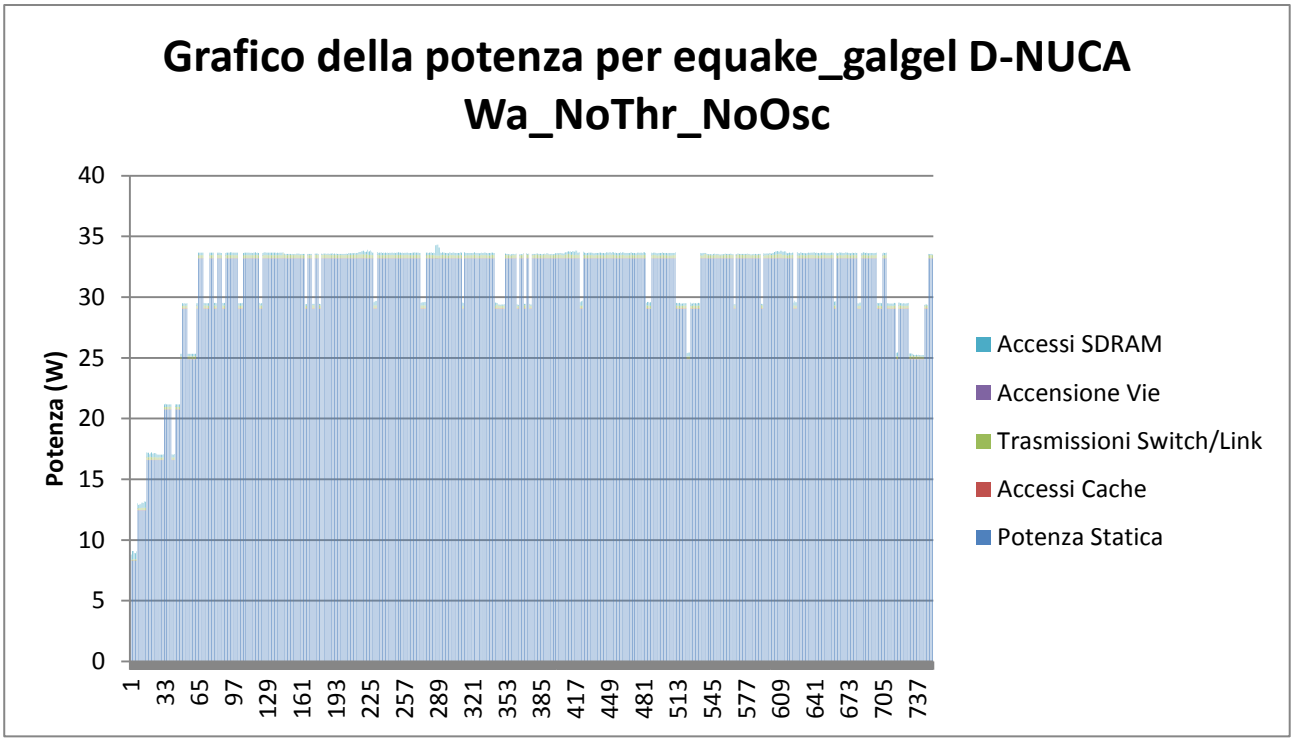


Grafico 43 : Grafico della potenza di equake_galgel D-NUCA Wa_NoThr_NoOsc

Cache	Valore
Plain	34,3244 W
Wa_AvgThr	33,85024 W
Wa_NoThr_NoOsc	34,32281 W

Tabella 25 : Peak Power per equake_galgel

2.1.3.1.9. *mgrid_equake*

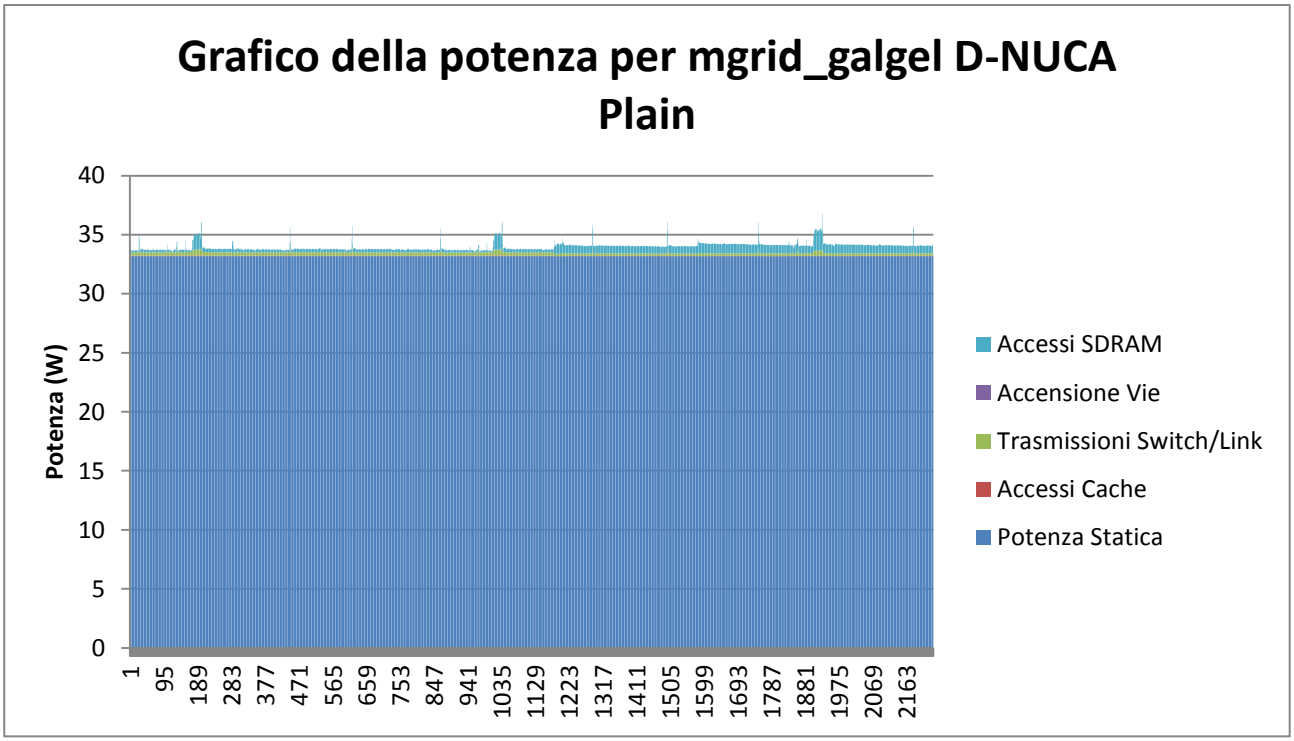


Grafico 44 : Grafico della potenza di mgrid_equake D-NUCA Plain

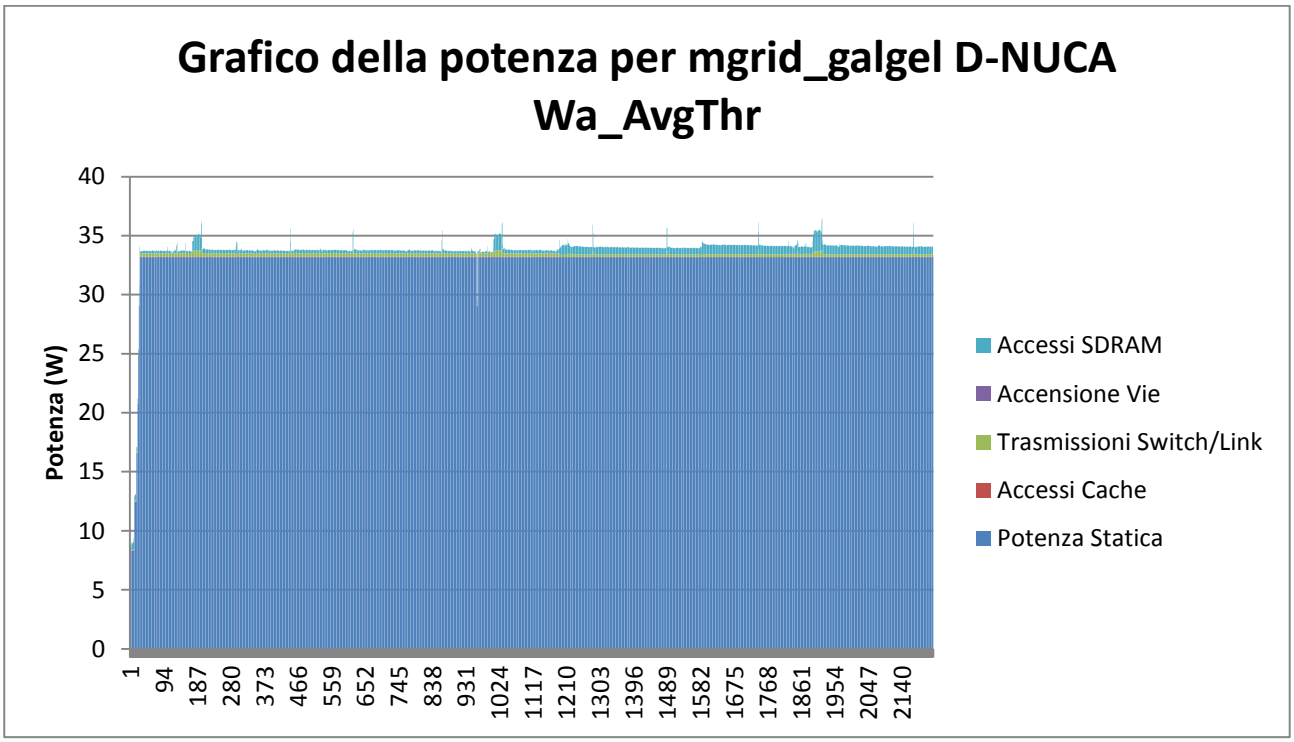


Grafico 45 : Grafico della potenza di mgrid_equake D-NUCA Wa_AvgThr

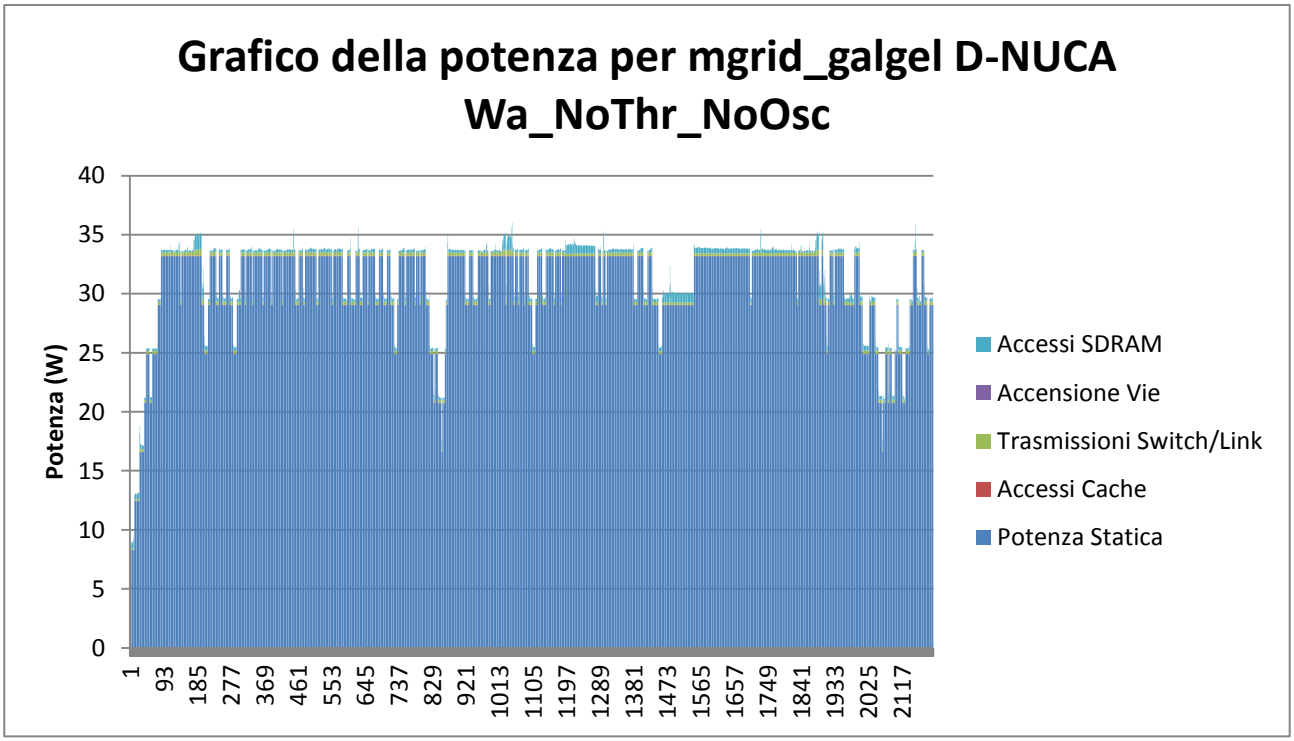


Grafico 46 : Grafico della potenza di mgrid_equake D-NUCA Wa_NoThr_NoOsc

Cache	Valore
Plain	36,83585 W
Wa_AvgThr	36,45835 W
Wa_NoThr_NoOsc	36,00734 W

Tabella 26 : Peak Power per mgrid_equake

2.1.4. Analisi dei risultati

Come si può notare dai grafici e dalle tabelle riportate in precedenza, si può notare che la potenza dinamica è irrisoria , se confrontata con la potenza statica. Tale contributo quasi infinitesimo rispetto al consumo di potenza effettivo , non inficia minimamente sul consumo globale che in futuro potrà considerarsi trascurabile rispetto al sempre più crescente consumo di potenza statica.

Per contrastare il sempre maggiore contributo statico alla potenza assorbita , in futuro si vedranno sempre più applicare , le tecniche di risparmio energetico basate sullo spegnimento o delle vie , o delle linee di cache.

L'algoritmo wayhalting analizzato in questa sede , riesce in alcuni casi a diminuire il consumo di potenza , con particolare riferimento alla configurazione senza soglie e senza oscillazioni Wa_NoThr_NoOsc. D'altro canto invece la configurazione basata su soglie euristiche Wa_Avg risulta meno perforante e , nella migliore delle ipotesi riesce solo ad eguagliare i risultati dell'altra caratteristica analizzata.

2.2. Confronto delle prestazioni tra cache S-NUCA e D-NUCA in modalità single core

In questo paragrafo ci occuperemo di analizzare le prestazioni delle cache S-NUCA , al fine di compararne con le prestazioni con le cache D-NUCA ,in modo da avere un quadro generale delle prestazioni delle varie tecnologie , valutando al tempo stesso le modifiche fatte al simulatore . In particolare il nostro obiettivo è anadare a scoprire se Le cache nuca riescono a mascherare l'aumento di latenza dovuto all'applicazione di tecnologie LSPD (leakage sensiing pipe detection). Tali tecnologie di riduzione del leakage impementate a livello hardware ,consentono una riduzione dei consumi notevole , andando però ad inficiare pesantemente nelle prestazioni ottenute. Per questo motico tali tecnologie vengono applicate in sistemi embedde /mobili in cui la riduzione del consumo di potenza è un fattore critico. L'analisi mirerà a valutare prinpalmente ,se tali tecnologie sono applicabili anche in campio dei processori ad alte prestazioni .Per effettuare l'analisi occorre effettuare una extended parameter exploration per poter capire come variano le prestazioni al variare delle tecnologie.

2.2.1. Architettura di riferimento

I nuclei simulati sono basati sulla microarchitettura Alpha 21264, in esecuzione auna frequenza di 3 GHz. , per poter comparare i risultati ottentuti con i precedenti. In sono riportate le caratteristiche della cache scelte per il confronto :

	D-NUCA/S-NUCA 16x8	**D-NUCA/S-NUCA 8x8**	**D-NUCA/S-NUCA 8x8**	**UCA**
Dimensione:	8 MB	8 MB	8 MB	8 MB
Dimensione Linea	64 B	64 B	64 B	64 B
N. banchi	128	64	64	1
Dimensione Banco	64 KB	128 KB	128 KB	8192 KB
N. righe	16	8	8	-
N. colonne	8	8	8	-
Associatività	Direct Mapped	Direct Mapped	Direct Mapped	Direct Mapped
Tecnologia	32 nm	32 nm	32 nm	32 nm

Tabella 27 : configurazioni S/NUCA D/NUCA

	S-NUCA 2x1	**S-NUCA 4x1**	**S-NUCA 4x2**
Dimensione:	8 MB	8 MB	8 MB
Dimensione Linea	64 B	64 B	64 B
N. banchi	2	4	8
Dimensione Banco	4096 KB	2048 KB	1024 KB
N. righe	2	4	4
N. colonne	1	1	2
Associatività	Direct Mapped	Direct Mapped	Direct Mapped
Tecnologia	32 nm	32 nm	32 nm

Tabella 28 : configurazioni S/NUCA

Abbiamo creato varie cache in Tabella 27 e Tabella 28 con la suite di modellizzazione CACTI 5.1 utilizzando varie librerie tecnologiche , in particolare .

- HP : librerie high performance sia per la circuiteria periferica che per le celle di memoria.
- LSP Cell : librerie high performance per la circuiteria periferica e low standby power per le celle di memoria
- LSP All : librerie low standby power sia per le celle di memoria che per la circuiteria periferica

Ciascuna libreria libreria è stata valutata con le seguenti ottimizzazioni di cacti

- Normal : ottimizzazione di default
- Opt Leakage : ottimizzazione operata da cacti per ottenere un risultato che soddisfi comunque i vincoli ma che riduca il più possibile il leakage

Ciascuna delle combianzioni prese in esame in precedenza è stata simulata utilizzando un accesso parallelo a data e tag o seriale.

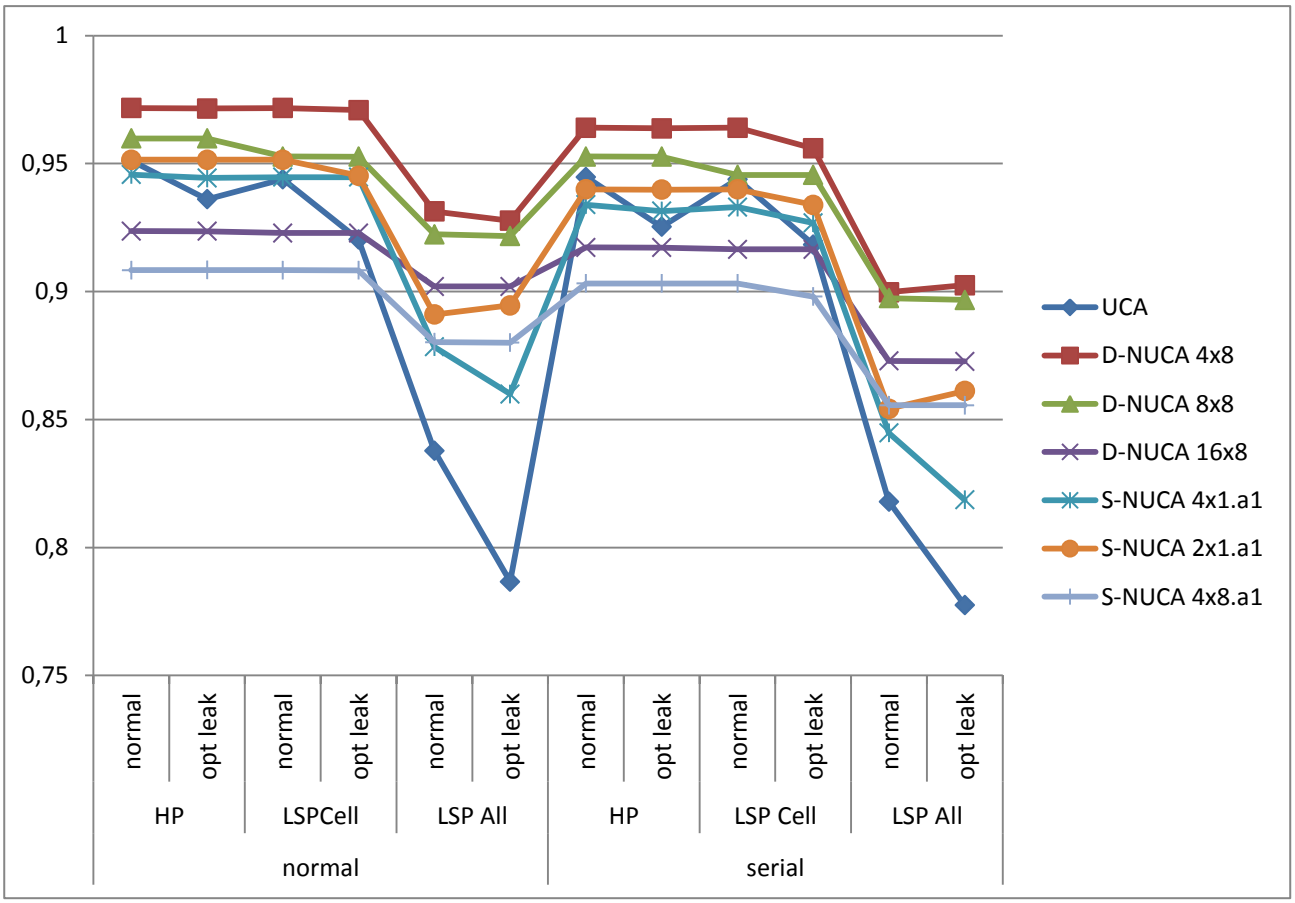
Andiamo adesso ad operare l'analisi delle prestazioni sia in modalità single core che in modalità dual core:

2.2.2. Confronto prestazioni D-NUCA/S-NUCA/UCA in single core

L'analisi è stata effettuata simulando i bechmark riportati in .

Benchmark	Suite	FFWD	RUN
Bzip2	SPECINT2000	744M	1.0B
Gcc	SPECINT2000	2.367B	300M
Mcf	SPECINT2000	5.0B	200M
Parser	SPECINT2000	3.709B	200M
Perlbmk	SPECINT2000	5.0B	200M
Twolf	SPECINT2000	511M	200M
Applu	SPECFP2000	267M	650M
Art	SPECFP2000	2.2B	200M
Equake	SPECFP2000	4.459M	200M
Galgel	SPECFP2000	4.0B	200M
Mesa	SPECFP2000	570M	200M
mgrid	SPECFP2000	550M	1.06B
Bt	NAS	800M	650M
Cg	NAS	600M	200M
Sp	NAS	2.5B	200M

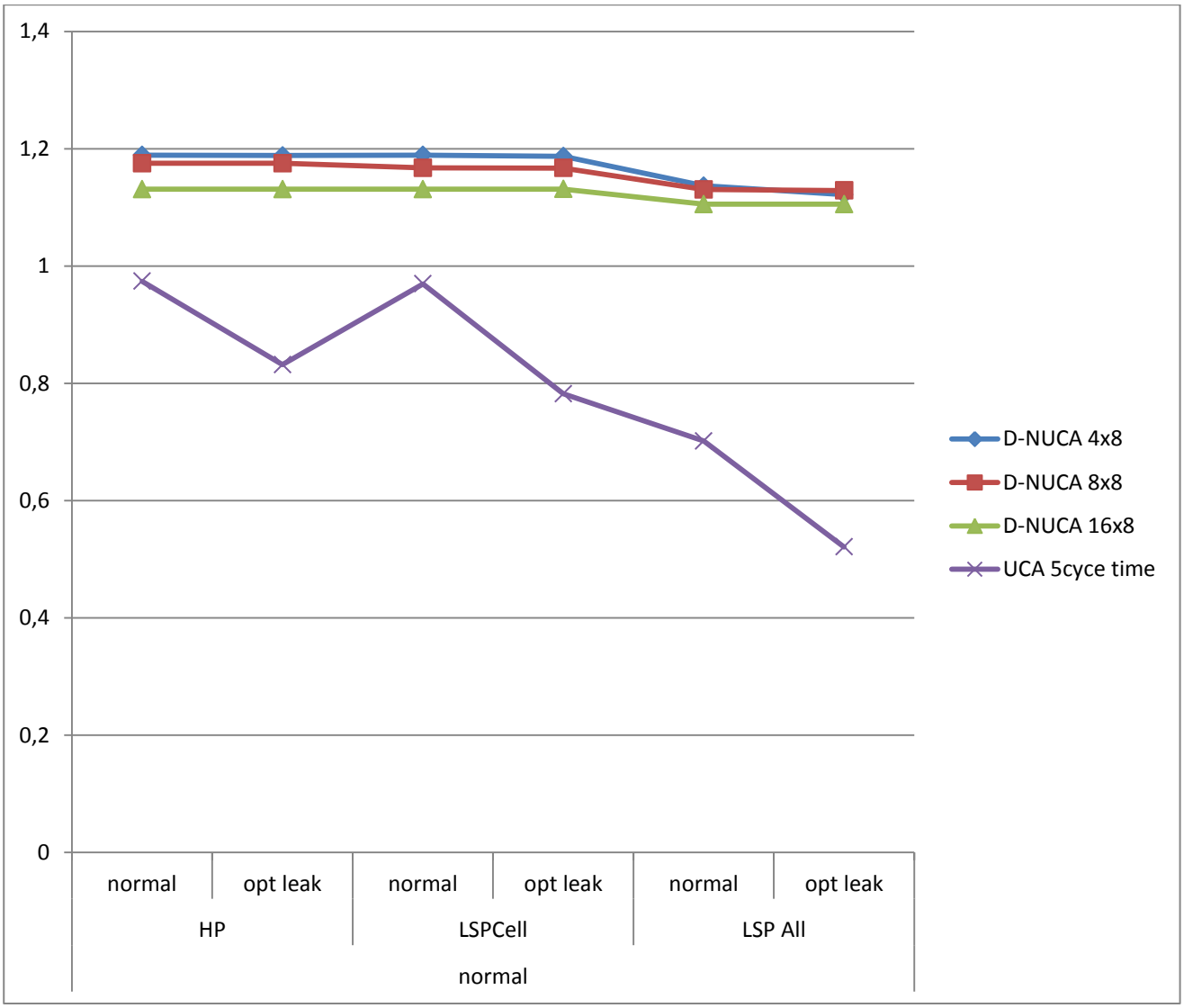
Abbiamo provveduto ad estrarre dai risultati delle simulazioni , IPC medio di ciascun bechmark , per ciascuna delle configurazioni di cache analizzate in precedenza. I risultati sono presentati in



2.2.3. Confronto prestazioni D-NUCA/S-NUCA/UCA in single core

Il confronto dual core è stato effettuato per le stesse tecnologie presentati in precedenza ma solo per la modalità di accesso normal . i bechmark schelti per le simulazioni sono riportati in

Il grafico riassuntivo delle prestazioni che tiene conto degli IPC di tutti e due core(media pesata) è riportato i figura



2.2.4. Analisi dei risultati

Come si nota dai grafici ,per quanto riguarda i dati in single core le prestazioni migliori si ottengono con le cache D-NUCA in particolare con la configurazione 4x8 . Tali risultati sono dovuti al meccanismo di migrazione dei dati nelle vie più vicine , che consente di ridurre i tempi di accesso riuscendo a sopperire i grandi tempi di accesso dovuti all'applicazione di tecnologie di risparmio energetico. Le cache s- nuca si comportano alla pari di alcune configurazioni di cache d-nuca, ma risentono pesantemente dei tempi di accesso lunghi per le tecnologie di tipo LSP. Per poter migliorare le prestazioni delle cache s-nuca una soluzione possibile consiste nell'aumentare

l’associatività dei singoli banchi. Per ottenere questo risultato occorre un ulteriore parameter exploration.

In ambiente dual core le cose migliorano drasticamente, e i gap che sembravano molto marcati nei risultati single core . Inoltre si note una riduzione molto minore delle prestazione con l’applicazione delle tecnologie LSP ALL . Tutte le tecnologie , comparate con le cache UCA , risultano in ogni caso avere prestazioni molto maggiori

BIBLIOGRAFIA

C. Kim, D.C. Burger, and S.W. Keckler: “NUCA: A Non-UniformCache Access Architecture for Wire-Delay Dominated On-Chip Caches”, *IEEE Micro*, pp. 99-107 November/December 2003

K. Flautner, et al: “Drowsy Caches: Simple Techniques for Reducing Leakage Power”, *Proc. Int. Symp. on Computer Architecture*, May 2002.

Z. Hu, et al: “Let caches decay: reducing leakage via exploitation of generational behaviour”, *ACM TOCS*, vol. 20(2), 161-190, 2002.

M. Powell, S. Yangh, B. Falsafi, K. Roy, N. Vijaykumar: “Gated-VDD:A Circuit Technique to Reduce Leakage in Deep-Submicron Cache Memories”, *Int. Symp. Low Power Electronics and Design*, July 2000.